UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática


# MONITORIZAÇÃO APLICACIONAL - SPUTNIK, CHECKLIST E GATEWAY ASTERISK

**Tiago Almeida Henriques**

**PROJECTO**

Mestrado em Engenharia Informática
Especialização em Arquitectura, Sistemas e Redes de Computadores

2012

UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática


# MONITORIZAÇÃO APLICACIONAL - SPUTNIK, CHECKLIST E GATEWAY ASTERISK

**Tiago Almeida Henriques**

Projecto orientado pelo Prof. Doutor Hélder Manuel Ferreira Coelho
e Mestre João Luis Ferreira de Amorim Sarmento Coelho

Mestrado em Engenharia Informática
Especialização em Arquitectura, Sistemas e Redes de Computadores

2012

# Agradecimentos

Em primeiro lugar quero agradecer aos meus pais por serem tão pacientes comigo e me terem dado todas as oportunidades para me poder formar e aos meus amigos por me aturarem durante este meu percurso.

À Olisipo e PTSI, respectivamente Adélia Carvalho e Miguel Carvalho pelo seu empenho para que eu pudesse fazer o PEI na empresa. Agradeço, também, aos colegas da minha actual equipa, Monitorização Aplicacional, especialmente ao Duarte Pimenta pela nossa amizade e pelas batalhas que já travámos juntos, tanto na faculdade como na PTSI. Nunca esquecendo o Miguel Santos, sendo o responsável pelas equipas de produção de 1ª e 2ª linha, soube sempre motivar-me mesmo nos maus momentos. Aos colegas da minha antiga equipa Supervisão TMN (actualmente Produção OA), que sempre me ajudaram e apoiaram não só como colegas mas também como amigos.

Quero, também, agradecer ao meu coordenador Professor Doutor Hélder Coelho, que me providenciou a visão para guiar a minha tese ao sucesso e ao meu co-coordenador João Sarmento pela sua disponibilidade para as minhas questões.

Finalmente, quero agradecer à FCUL por todo o seu empenho na minha formação pessoal e académica.

*Dedicado aos meus pais Maria do Carmo e António Henriques.*

# Resumo

Todos os Sistemas de Informação encontram-se susceptíveis a falhas, sejam elas humanas, de infra-estrutura ou aplicacionais e, portanto, necessitam de constante monitorização para não haver quebras de serviço que afectem o negócio. Esta necessidade de monitorizar e de actuar o mais rapidamente possível sobre sistemas críticos e o facto de trabalhar numa área em que a função principal é monitorizar os sistemas aplicacionais e infra-estruturais da PT, abriu-me portas para encontrar novas soluções que ajudem as equipas de operação.

O *Sputnik* é uma plataforma de representação gráfica e intuitiva de monitorizações infra-estruturais e aplicacionais, permitindo representar circuitos, tabelas, gráficos ou por exemplo, verificando a disponibilidade de um servidor ou o acumular de registos numa tabela. Com esta plataforma os problemas são detectados/visualizados em *real-time* e resolvidos ou escalados de forma rápida e eficiente.

A *Checklist* é uma plataforma centralizada onde o conhecimento e a informação estão organizados/estruturados em CIs (*Configuration Items*) e relações. Pretende ser a base de consulta para qualquer tipo de informação relevante, incluindo relações entre os vários *CIs* existentes, para a DE/OA.

A *Gateway Asterisk* é uma plataforma piloto (*proof-of-concept*) destinada a despoletar chamadas telefónicas automáticas substituindo a necessidade de acção humana, agilizando assim o processo de escalamento. No contexto do meu PEI as chamadas são destinadas a alertar as equipas responsáveis em caso de falha dos *CIs*.

Neste PEI pretendi não só melhorar/desenvolver as plataformas de alarmística sobre os sistemas da PTSI mas também melhorar os meus conhecimentos técnicos, de gestão e dar um contributo efectivo sobre os processos de monitorização e alarmística minimizando a acção humana e auxiliando-a sempre que necessário.

**Palavras-chave:** monitorização, falha, sistema, alarme, *CI*

# Abstract

All information systems are prone to failures, whether they're human, infrastructural or applicational, therefore requiring constant monitoring in order to prevent service unavailability affecting business.

*Sputnik* is a platform for graphical and intuitive representation of infrastructural and applicational monitoring, allowing to represent circuits, tables, graphs or for example checking the availability of a server or the accumulation of records in a table. This platform allows real-time problem detection/visualization and supports it's resolution/escalation in a efficient way.

The *Checklist* is a centralized platform where knowledge and information are organized/structured in CI's (Configuration Items) and relationships between them. Intended as a framework for searching any type of relevant information, including relationships between the existing CIs.

*Asterisk Gateway* is a pilot platform (proof-of-concept), designed to trigger automatic phone calls replacing the need for human action, thus speeding the process of scaling. In the context of this PEI, calls are triggered when a CI falis, alerting the respective support team.

In this PEI I intended not only, to improve/develop alarmistic platforms over the PTSI systems but also to improve my technical knowledge, management and effective input on the process of monitoring and alarms, minimizing human action and helping them when necessary.

**Keywords:** monitoting, failure, system, alarm, CI

# Conteúdo

# Lista de Figuras

# Lista de Tabelas

# Capítulo 1

# Introdução

O meu Projecto de Engenharia Informática (PEI) foi proposto pela Olisipo SA (empresa de renome em consultoria, *outsourcing*, inovação e formação) e está a ser realizado na PTSI (Portugal Telecom Sistemas de Informação), que faz parte do grupo PT (maior grupo de telecomunicações em Portugal). Dentro da PTSI estou inserido dentro da área DE/OA (Direcção de Exploração/Operação Aplicacional) numa pequena equipa de desenvolvimento orientada para os serviços de Monitorização Aplicacional. Esta é a primeira equipa de desenvolvimento dentro da OA, resultante da experiência nesta área angariada ao longo de vários anos de supervisão de sistemas de informação PTSI.

Ao longo de três anos na área de OA efectuei um percurso em crescendo, cumprindo actualmente também funções de 2ª linha que passam por componentes de gestão, como acompanhamento de equipas de operação 24x7, *coaching*, *backup* técnico de identificação e resolução de incidentes nos sistemas aplicacionais, formação de novos elementos, etc.

O principal objectivo da OA é a monitorização aplicacional do grupo PT e a automatização de procedimentos procurando a optimização do esforço, através de tecnologias e competências ao serviço da equipa. Esta automatização garante uma resposta mais rápida das equipas da OA, diminuindo o impacto no negócio e, consequentemente, no cliente final.

É dentro deste contexto que o meu PEI se enquadra, através de implementação de monitorizações, centralização de toda a informação envolvente sobre *CIs* (*Configuration Items*) aplicacionais numa só plataforma. Através desta, é possível interligar as várias aplicações de monitorização desenvolvidas na OA, disponibilizando assim, como um serviço à organização, toda a informação pertinente em cada momento, dinâmica e automaticamente.

## 1.1 Motivação

O Projecto de Engenharia Informática (PEI) garante-me a oportunidade de combinar uma formação mais avançada e desenvolver-me tanto profissional como pessoalmente.

Para além de ser uma meta académica exigente, é acima de tudo um desafio às minhas capacidades de aprendizagem e aos meus conhecimentos, permitindo-me colocar os meus objectivos em níveis cada vez mais altos.

A possibilidade do projecto ser realizado numa empresa com a dimensão da PTSI (através da Olisipo) veio dar-me toda a liberdade para crescer, ao lidar com pessoas diferentes e os vários desafios que me são colocados, sejam a nível técnico ou a nível de gestão de equipas, problemas críticos, situações de alguma forma inesperadas que requerem prontidão e lucidez na sua análise, resolução e escalamento.

Dentro deste quadro foi-me proposto a implementação de duas plataformas e um piloto para realizar chamadas automáticas, respectivamente o *Sputnik*, a *Checklist* e a *Gateway Asterisk*.

Na plataforma *Sputnik*, que não está a ser implementada mas sim continuada estou responsável pela sua manutenção e evolução da consola de gestão das monitorizações, assim como o desenvolvimento de uma aplicação *Web* para criação de novas instâncias *Sputnik* e um motor dedicado para gerar alarmes se os valores obtidos das monitorizações assim o exigirem.

A plataforma *Checklist* foi implementada de raiz, incluindo elaboração e implementação do modelo de dados, aplicação *Web* para gestão e pesquisa de *CIs* aplicacionais e *Webservices* para integração com as outras plataformas de monitorização da OA.

A concepção de um *proof-of-concept Asterisk* permitiu o despoletar de chamadas telefónicas automáticas em caso de alarme. Nesta plataforma foi necessário o desenvolvimento da arquitectura do piloto, implementação e invocação/integração com outras aplicações. Esta automatização permitiu a diminuição do peso que os contactos telefónicos têm na Produção OA.

## 1.2 Objectivos

### 1.2.1 Projecto Sputnik

A ideia original do projecto *Sputnik* nasceu da necessidade de visualizar de forma global e integrada o circuito de recargas de cartões da TMN, dada a sua criticidade e funcionamento *real-time* com visibilidade directa no cliente final. Este circuito tem origem nos sistemas SIBS (ATM ou caixas Multibanco) e também vários parceiros que prestam este serviço, como por exemplo a *Payshop*. Ao longo do tempo foram desenvolvidas várias camadas de alarmística sobre este circuito, não existindo porém uma visão integrada do seu estado num determinado momento.

Sendo a missão da Produção OA garantir que qualquer problema, particularmente neste circuito, é resolvido ou escalado num curtíssimo espaço de tempo (na ordem de poucos minutos), a ideia do *Sputnik* permitiu ter de forma praticamente imediata a identificação do problema, dos impactos e consequentemente uma visão rápida da solução a aplicar,

bem como a recolha de dados ao longo do tempo sobre o circuito de recargas que permitem análises de performance aplicacional, cumprimento de *SLAs*, entre outros. O acesso *Web* permite a todos os utilizadores autorizador um fácil e rápido acesso ao estado do circuito.

Esta nova visão do circuito de recargas foi também um contributo ao nível de gestão e coordenação das equipas de operação, que assim passaram a ter (sem muitos detalhes técnicos) o estado do circuito e detecção em *real-time* do ponto de falha.

Com a grande visão que esta plataforma já tem dentro da empresa, o principal objectivo da minha acção foi:

- Implementação de uma forma automática de *rollout* de novas instâncias *Sputnik*, que permitam visões idênticas sobre outros sistemas críticos que careçam também de visão integrada
- Implementação de um motor dedicado que recebe os valores das monitorizações e gera os respectivos alarmes, em caso de necessidade
- Integração com plataforma OA de *deploy* de monitorizações *out-of-the-box*, que permitem facilmente a implementação de monitorizações standard (numero de ficheiros numa directoria, volume de registos numa tabela, tamanho de *logfiles*, etc)
- Implementação de uma consola *Web* de gestão de parametrizações e configurações de cada instância do *Sputnik*
- Criação e manutenção de documentação da aplicação

### 1.2.2 Gateway Asterisk

Inicialmente, o objectivo do projecto *Gateway Asterisk* foi a elaboração de um piloto (*proof-of-concept*) que permitisse ser invocado perante uma condição de alarme e desencadear de forma automática um telefonema para, por exemplo, uma prevenção de suporte técnico, indicando a existência de alarmes por verificar e corrigir.

Com o decorrer do PEI, o piloto foi terminado e consegui fazer com que uma equipa de suporte técnico ajudasse na execução de testes reais. Os testes foram um sucesso e o objectivo passou a ser um projecto sólido pronto para ser utilizado no *report* de diversos alarmes.

### 1.2.3 Projecto Checklist

A *Checklist* é principalmente uma base centralizada de conhecimento e informação, interna, relevante para toda a OA, ou seja, pretende-se com esta plataforma centralizar/orientar toda a informação sobre equipas e elementos, sistemas e responsáveis e ainda problemas conhecidos e as suas resoluções/escalamentos. Existe uma associação lógica

dos vários contactos, com máquinas e/ou aplicações, podendo estes ser responsáveis ou interessados pelos mesmos. Esta lógica permite não só as associações indicadas mas também qualquer outro caso de necessidade.

Para além dos contactos, pretende-se também que a *Checklist* sirva de repositório de informação sobre problemas com resolução e/ou escalamento comuns às várias equipas que compõem a OA. Serve assim como catálogo de contactos e problemas existentes das equipas de produção da OA.

Esta solução já tem uma integração interna com várias ferramentas desenvolvidas pela OA, dado que, de futuro, todas as ferramentas deverão aceder à *Checklist* quando pretenderem alguma informação necessária, seja um contacto/e-mail de uma equipa ou informação sobre um sistema ou aplicação interna.

É garantida também uma integração externa com a *CMDB* (*Configuration Management Database*) federada da PT. Não se pretende substituir a *CMDB* mas sim melhorar e automatizar a consulta à sua informação por parte da OA, sendo por isso uma camada complementar, paralela e independente da *CMDB*.

Esta aplicação é necessária devido à dificuldade em adicionar informação à *CMDB* existente, pois é uma aplicação fechada e gerida por equipas fora da OA.

A *Checklist* permite, portanto, que possamos gerir a informação da *CMDB* de forma independente e disponibilizá-la à OA de maneira útil e simples, diminuindo assim a entropia que, por vezes, é causada na pesquisa da informação pretendida.

### 1.2.4 Gestão

Como elemento de 2ª linha tenho a responsabilidade de acompanhar as equipas de 1ª linha, ajudando-os em problemas reais e sendo *backup* 24x7 para qualquer situação mais ou menos grave e, onde existam quaisquer dúvidas. Dei formação aos novos elementos de 1ª linha, preparando-os para os desafios futuros.

Na componente de formações dei formações mais técnicas como *shellscript*, *C#* e *SQL*.

Pretende-se também automatizar vários processos que as equipas de 1ª linha executam manualmente, poupando assim tempo e recursos, podendo estes serem alocados em locais onde são mais necessários.

# Capítulo 2

# Planeamento

Um bom planeamento é um ponto fulcral na maioria dos projectos e a sua importância reflecte-se quando muitos correm mal devido a um mau planeamento ou ausência do mesmo.

Neste projecto, o planeamento serviu para me manter focado nos objectivos a que me propus e foi uma forma de controlar qualquer tipo de atraso ou adiantamento do plano. Normalmente, um desvio detectado atempadamente consegue ser facilmente mitigado ao contrário do que é só detectado quando nos deparamos com as suas consequências.

Inicialmente concentrei-me na definição dos objectivos macro/*milestones* e à medida que o projecto se vai desenvolvendo, aplico o método de *Rolling Wave* [7] para detalhar as tarefas com maior granularidade.

Com o avançar do projecto é feita uma comparação entre o planeamento inicial e o trabalho efectuado.

Ao fazer a preparação do planeamento para este projecto deparei-me com uma citação que enfatiza a importância de um bom planeamento:

*"Executar um projecto sem planeamento é como ir para uma terra estranha sem mapa."* [16]

## 2.1 Rolling Wave

O *Rolling Wave* é um processo de planeamento por etapas ("ondas") à medida que o projecto se torna mais claro e definido.

Dependendo do projecto (extensão e complexidade) podemos planear algumas semanas ou mesmo alguns meses em avanço com uma quantidade razoável de clareza. Este planeamento envolve a criação de uma estrutura *Work Breakdown* detalhada e bem definida para esse período de clareza e assinalando apenas as *milestones* para o resto do projecto.

Este planeamento é aplicado no meu PEI através da aplicação do nível de detalhe

adequado ao horizonte de um planeamento confortável. Para além desse horizonte, são definidas as actividades ou fases do projecto num nível genérico, reflectindo o aumento gradual do nível de incerteza. O horizonte de planeamento move-se para diante conforme o projecto vai progredindo. As actividades genéricas que são inicialmente vagas, vão sendo detalhadas quando se aproximam das respectivas datas.

## 2.2 Milestones

Procurou definir-se as *milestones* do projecto com uma periodicidade mensal, garantindo uma distribuição de esforço equilibrada. A data limite é o final do respectivo mês.

| **Planeamento inicial** | |
|---|---|
| **Mês** | **Plano** |
| Outubro | Contextualização Projecto *Sputnik* e *Checklist*<br>Formação em C# .NET e MVC |
| Novembro | Documentação do Projecto *Sputnik*<br>Manutenção Evolutiva do *Sputnik* |
| Dezembro | Consola de Gestão *Sputnik*<br>Interface Instalação *Sputnik*<br>Relatório Preliminar |
| Janeiro | Análise de Plataformas de Agentes<br>Segmentação do motor *Sputnik* / interface Web |
| Fevereiro | Protótipo *Sputnik*<br>Análise de Requisitos *Sputnik* Agent |
| Março | Protótipo *Sputnik* Agent<br>*Gateway Asterisk*<br>Documentação *Sputnik* |
| Abril | Análise de Requisitos *Checklist*<br>Projecto *Checklist* |
| Maio | Projecto *Checklist*<br>Integrações |
| Junho | Documentação Projecto *Checklist*<br>Relatório Final |

Tabela 2.1: Planeamento Inicial

## 2.3 Planeado vs Realizado

Naturalmente, todos os projectos podem sofrer atrasos e alterações no seu percurso e, o meu PEI não fugiu à regra, sendo que o planeamento teve de ser ligeiramente alterado, não só devido a alguns atrasos mas também por modificações no próprio projecto.

Como fiz o PEI em paralelo com o trabalho que já desenvolvia na PTSI, tendo outro tipo de responsabilidades e de funções dentro da minha área, como o acompanhamento

| Trabalho efectuado | | |
|---|---|---|
| **Mês** | **Projecto** | **Milestone** |
| **Outubro 2011** | – | Formação em .NET (C# e MVC) |
| | Sputnik | Contextualização Projecto |
| **Novembro 2011** | Sputnik | Documentação do Projecto |
| | Sputnik | Manutenção Evolutiva |
| **Dezembro 2011** | Sputnik | Interface *Web* |
| | Sputnik | Interface de Gestão |
| **Janeiro 2012** | Sputnik | Interface de Gestão |
| | Sputnik | Interface de Instalação (*Rollout*) |
| **Fevereiro 2012** | Sputnik | Interface de Instalação (*Rollout*) |
| | Sputnik | Inicio do *Sputnik WebService* |
| | – | Relatório Preliminar |
| **Março 2012** | Sputnik | *Sputnik WebService* (Integração com *Mon-IT*) |
| | Sputnik | Documentação *Sputnik* |
| | Asterisk | Contextualização Projecto |
| **Abril 2012** | Asterisk | Desenvolvimento da plataforma |
| | Checklist | Análise de Requisitos |
| | Checklist | Desenvolvimento do Projecto |
| **Maio 2012** | Checklist | Desenvolvimento do Projecto |
| | Checklist | Integrações |
| **Junho 2012** | Checklist | Projecto *Checklist* |
| | Checklist | Documentação Projecto *Checklist* |
| | – | Relatório Final |

Tabela 2.2: *Milestones* do trabalho efectuado

das equipas de 1ª linha, instalação de *software*, entre outros. Este conjunto de funções extra PEI provocou alguns atrasos na realização do mesmo.

Apesar dos atrasos, a alteração de integrar o *Sputnik* com o *Mon-IT*, em vez de criar o *Sputnik Agent*, efectuada em relação ao planeamento inicial, acabou por compensá-los.

Por outro lado a experiência acumulada e o planeamento realizado permitiram que não fossem cometidos alguns erros e queimadas algumas etapas.

# Capítulo 3

# Projecto Sputnik

## 3.1 Contextualização

Na sequência do que já foi referido no ponto (secção 1.2.1), existem processos responsáveis pela colecta dos valores para as monitorizações que se encontram criados em cada máquina e são específicos para cada tipo de sistema operativo (*HP-UX*, *Solaris*, *Tru64*, *Windows*, etc).

Devido à aplicação já estar aplicada em vários domínios, necessitei de adquirir um conjunto de conhecimentos alargados sobre a mesma. Para obter este conhecimento pretendo criar/actualizar documentação e ser o responsável pela manutenção correctiva e evolutiva da aplicação.

Os processos de colecta, doravante denominados "**sondas**", são criados localmente nas máquinas onde os serviços e processos a monitorizar residem. A grande parte desses processos correm em plataformas *Unix* (*Tru64*, *HP-UX* e *Solaris*), pelo que também a maioria das sondas foram implementadas em *shell script*, num único processo *multitask* por cada máquina que compõe o fluxo de recargas da TMN, ou seja, temos três processos a efectuar a colecta do estado de todo o circuito de recargas, com uma relação processo-sonda "um para muitos". Estes valores são colectados nos diversos pontos de controlo (tabelas, directorias, processos, etc), sobre os diferentes sistemas operativos e são passados ao *Sputnik* através de um *Webservice* desenvolvido para o efeito.

No entanto, o *Sputnik* original, que era responsável pela monitorização do processo de recargas, dadas as suas necessidades e tecnologias envolvidas, cada *script* passava os valores colectados através da invocação de um *SP* (*Stored Procedure*).

Cada *script* corre com um intervalo temporal configurável caso a caso, através do *scheduler*, efectuando verificações de dois tipos: **temporais** ou **numéricas**. Uma verificação temporal pode ser, por exemplo, o tempo de inactividade entre duas execuções de um determinado processo. Uma verificação numérica pode ser, por exemplo, o número de ficheiros numa determinada directoria ou o número de registos numa determinada tabela. Para cada um dos casos existem parametrizações específicas que permitem tratar os

valores recebidos e representar os mesmos na aplicação *Web*.

Esta plataforma envolve tecnologias, como bases de dados *Informix* e *SQL Server*, aplicações *Web* em *ASP.NET MVC 2* e *ASP.NET MVC 3*, *webservices* em *C#* recebendo e respondendo a pedidos em *XML*, *LINQ* para criar pedidos *SQL* a BDs em *SQL Server* ou *scripts* para as monitorizações em *shell script*, *perl* e *vbs* (de várias fontes pois podemos receber valores de aplicações exteriores), entre outras. A componente *Informix*, além da estrutura de tabelas que suportava os dados, suportava, também, o *SP* responsável por todas as comunicações com o *Sputnik*. Este *SP* garantia robustez dos dados, tendo em conta as parametrizações existentes, as chaves e regras de integridade, bem como a pertinência dos dados consoante as operações.

Após estar contextualizado com o *Sputnik* e, para me envolver neste projecto, comecei por actualizar documentação antiga, criar nova documentação e verificar, naquilo que já existia, o que era necessário evoluir/implementar.

Ao sentir-me devidamente contextualizado com o projecto que existia, passei a utilizar uma BD *SQL Server*, evolui o modelo de dados, dei inicio ao processo de evolução da Consola de Gestão, ao novo interface de instalação (processo de *Rollout*) e à introdução de *webservices* para integração com as outras aplicações (que até à altura não existiam).

## 3.2 Modelo de Dados

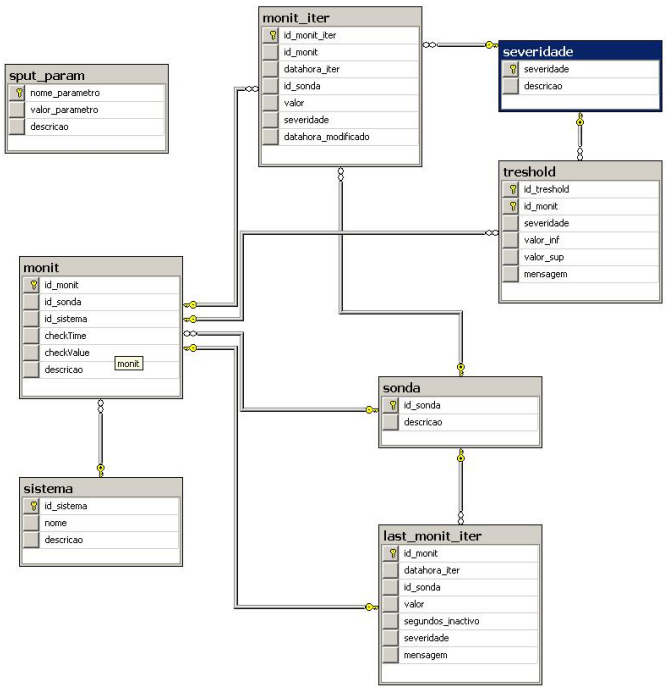


Figura 3.1: Sputnik - Modelo de Dados Beta

Quando comecei a contextualizar-me com este projecto, fui estudar o modelo de da-

dos 3.1 existente e, após uma análise cuidada, percebi que este modelo poderia ser refinado de forma a optimizar a sua utilização nas consultas efectuadas.

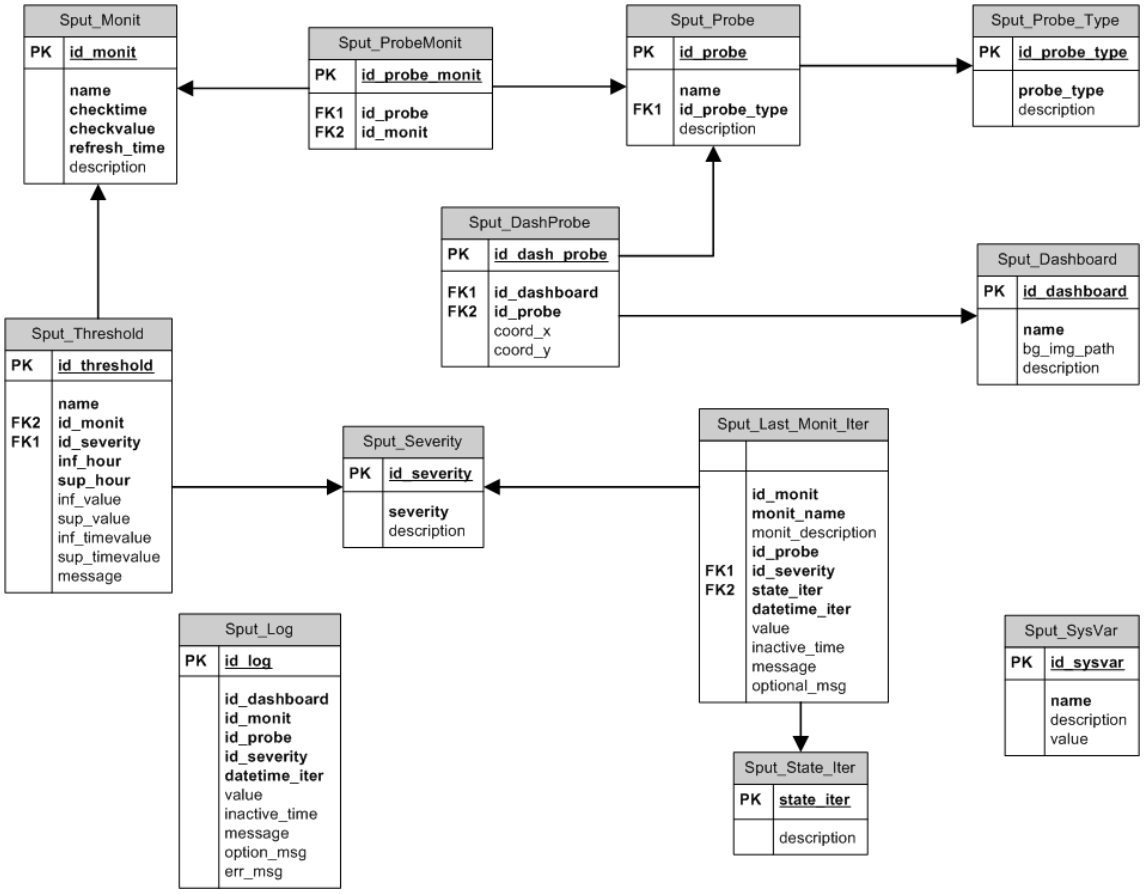


Figura 3.2: Sputnik - Modelo de Dados

Este novo modelo de dados 3.2 foi desenhado utilizando a lógica existente, contudo decidi normalizar o modelo e acabei por alterar praticamente tudo, mantendo a base.

A primeira alteração que efectuei foi a criação da tabela *Sput_Dashboard* para ter um registo do circuito a representar, incluindo a imagem de fundo a ser utilizada. Um *dashboard* pode conter várias sondas (*Sput_Probe*), que por sua vez podem ser de vários tipos (*Sput_Probe_Type*). Para esta relação criei a tabela *Sput_DashProbe*, também utilizada para guardar as coordenadas da imagem, onde as sondas deverão aparecer (isto é possível, pois cada sonda só pode estar associada a um *dashboard*).

Conforme foi dito anteriormente, as sondas podem ter vários tipos, neste momento podem ser *LEDs*, valores em tabelas ou valores em mensagens opcionais. À semelhança dos *dashboards*, as sondas podem conter várias monitorizações (*Sput_Monit*) e essa relação encontra-se na *Sput_ProbeMonit*.

Tal como anteriormente, uma monitorização só pode ser de dois tipos, *numérica* ou *temporal*, mas acrescentei o campo *refresh_time*, que até aqui era definido na tabela *sput_param*, desta forma cada monitorização pode ser actualizada de forma independente.

As monitorizações têm associados *thresholds* (*Sput_Threshold*) que têm uma severidade associada, um intervalo temporal onde é despoletado um alarme com a respectiva severidade e um intervalo de valores (numéricos ou temporais) para verificar se as monitorizações se encontram no intervalo correcto.

A tabela *Sput_Last_Monit_Iter* é a tabela que contém os valores das monitorizações (inseridos via *WebService*). Nesta tabela para além do *id*, nome e descrição da monitorização e *id* da sonda, também contém o estado da iteração (activa ou inactiva - *Sput_State_Iter*), a hora a que o valor foi recolhido, o valor (numérico - *value* ou temporal *inactive_time*) e duas mensagens, uma delas opcional. Contém também a severidade da iteração, ou seja, é verificado se o valor recolhido está no intervalo de algum *threshold* e a iteração da monitorização fica com a severidade do *threshold*.

Sempre que chega um valor para ser inserido na *Sput_Last_Monit_Iter*, se já existir alguma entrada para essa monitorização, o mesmo é apagado, colocado na tabela *Sput_Log* e o novo valor é inserido.

Finalmente, a tabela *Sput_SysVar* foi criada para conter variáveis de sistema, normalmente utilizadas em processos de *HouseKeeping*, desta forma podemos alterar o valor das variáveis via *Web*. Resumidamente, o objectivo é garantir que a plataforma é altamente parametrizável.

## 3.3 Motor dedicado

O motor do *Sputnik* é uma aplicação que é invocada na sequência de uma monitorização (recolha de valor) e é responsável por aplicar a criticidade que esse valor tem no contexto da monitorização, reportar o alarme (caso se aplique) e actualizar os respectivos *dashboards*. Garante:

- Robustez e validade nos dados parametrizados de acordo com as regras definidas (*constraints*, *keys*, relações, etc);
- Interpretação dos pedidos efectuados pelo *webservice*;
- Resposta aos pedidos efectuados pelo *webservice*;
- Manutenção de histórico de todas as operações efectuadas;
- Registo de erros e excepções;
- *Housekeeping* da BD, de acordo com as parametrizações tempo de *backlog* a manter;

Inicialmente, o motor foi desenhado para que a actualização das sondas fosse sempre feita através de uma consulta directa à base de dados. Isto tornava a actualização das

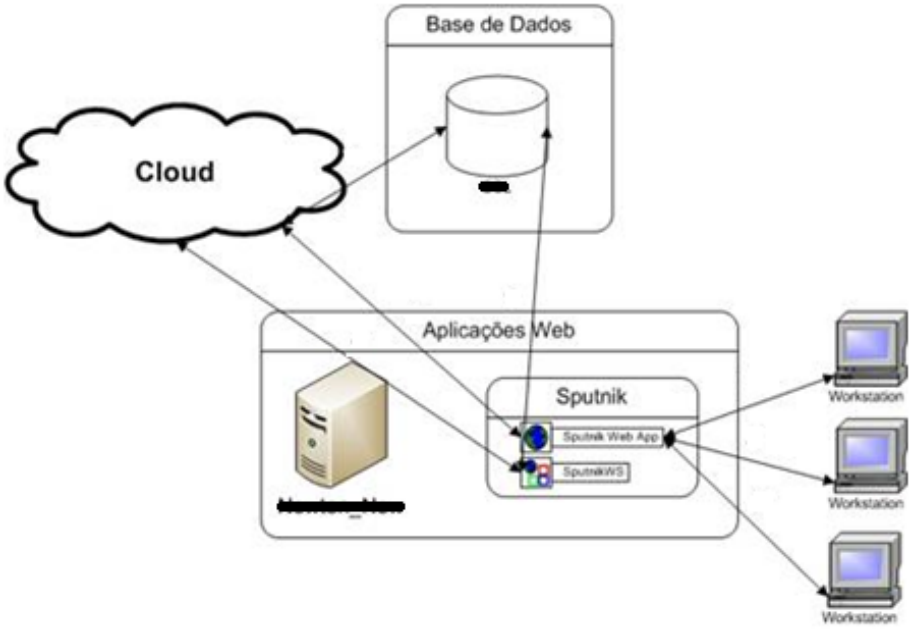


Figura 3.3: Sputnik - Motor dedicado

sondas e, consequentemente, dos *dashboards* bastante lenta e caso existissem *locks* nas tabelas, as sondas não eram actualizadas.

Esta forma de actualização era demasiado lenta e pesada para uma base de dados acedida por centenas de utilizadores dentro da empresa. Este problema também era resultado da utilização de uma BD *Informix* e a actualização dos valores na *Sput_Last_Monit_Iter* ser feita via *SP*. Ao utilizar uma BD *SQL Server* num servidor *Windows* actual, estes problemas já não são vistos com tanta gravidade.

O motor passou a ser o próprio *Web Service*, que ao ser invocado pelos processos de colecta de valores, consulta os *thresholds* da respectiva monitorização e insere o valor na *Sput_Last_Monit_Iter* com a severidade e mensagens respectivas. A aplicação *Web* ficou responsável pela verificação da inactividade das monitorizações, ou seja, verifica o *refresh_time* da monitorização e caso tenha sido ultrapassado coloca-a como inactiva.

Os *logs* que estão a ser guardados, irão servir para implementar uma funcionalidade futura, talvez numa aplicação à parte, nomeadamente o *threshold* dinâmico, o qual consiste em, através do histórico das monitorizações, estabelecer uma relação entre monitorização, valor e a hora em que o mesmo foi lido. Com estes dados é possível verificar automaticamente se é um problema ou não, isto é, se houve uma acção automática ou humana para resolver a situação. Para além do dinamismo dos alarmes, esta funcionalidade também é muito relevante para saber quais os alarmes que são necessários e os que podemos considerar spam. Desta forma podemos suprimir alarmes desactualizados de forma rápida e com um mínimo de esforço.

É pretendido ainda efectuar a supressão de alarmes em duas outras situações:

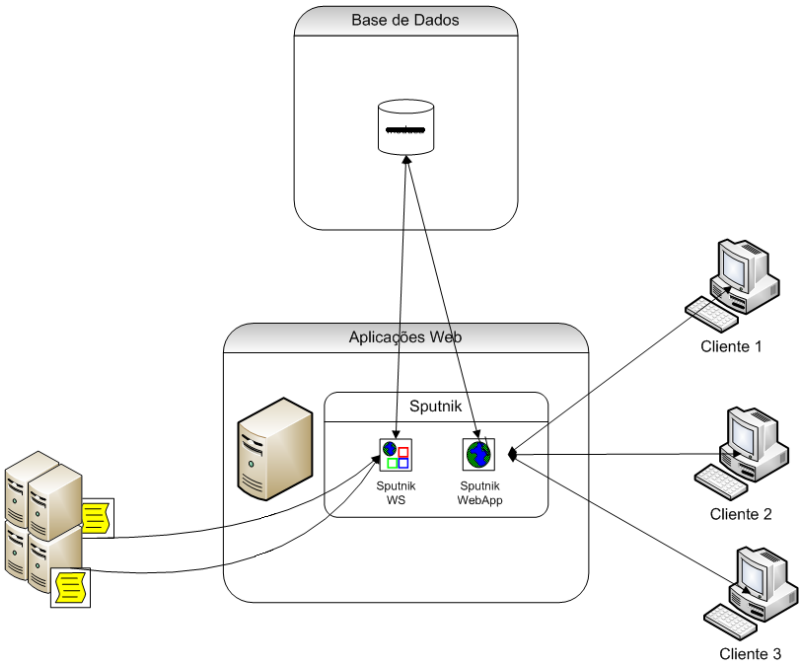


Figura 3.4: Sputnik - Motor (lógica actual)

- Intervenção planeada: Integração com o Medusa (ferramenta de gestão de indisponibilidades), que mapeia ao *CI* o período de indisponibilidade que o mesmo terá devido à intervenção e que disponibiliza essa informação para consulta via *WS*.

- Indisponibilidade não planeada já reportada: Não é necessário repetir notificações dos alarmes já reportados (especialmente *sms* e *emails*).

## 3.4 Aplicação Web

A aplicação *Web* baseia-se nos valores colectados pelos processos locais para apresentar o resultado em forma de sondas.

A primeira versão da aplicação *Web* (*Sputnik*), desenvolvida para o circuito de recargas da TMN, englobava todo o circuito do *Sputnik*. Continha a consulta na base de dados de todos os valores da última iteração da sonda. Consoante os valores obtidos, caso a caso, efectuava a representação gráfica das sondas.

Conforme é mostrado na imagem 3.5, estão representadas todas as máquinas do circuito de recargas da TMN e cada *led* é uma sonda que contém um conjunto de monitorizações aos vários processos responsáveis por processar os ficheiros de recargas dos clientes. O esquema 3.5 não se encontra completo devido a existir informação confidencial.

Com o objectivo de apresentar a informação da forma mais completa possível, permitindo uma rápida análise aos técnicos, bem como uma fácil percepção a todos os colaboradores com acesso à aplicação, a mesma foi implementada de forma a permitir alguma

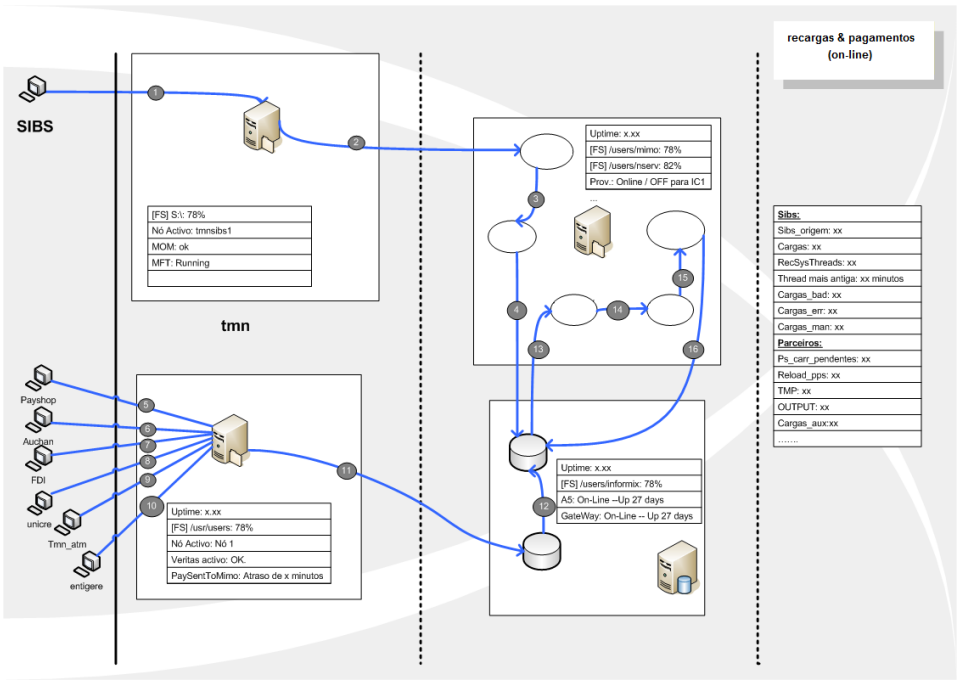


Figura 3.5: Sputnik - Circuito Recargas TMN

interactividade entre o utilizador e a aplicação, com o objectivo de se obter informação sucinta sobre o estado de cada sonda. Colocando o cursor do rato sobre cada uma das sondas, incluindo as tabelas anexas com os valores correspondentes à performance das máquinas e processos particulares, a aplicação apresentará informação sobre a mesma.

Tome-se o exemplo da sonda 4, colocando o rato sobre a mesma quando se encontra no estado verde (OK), irá aparecer o seguinte quadro:

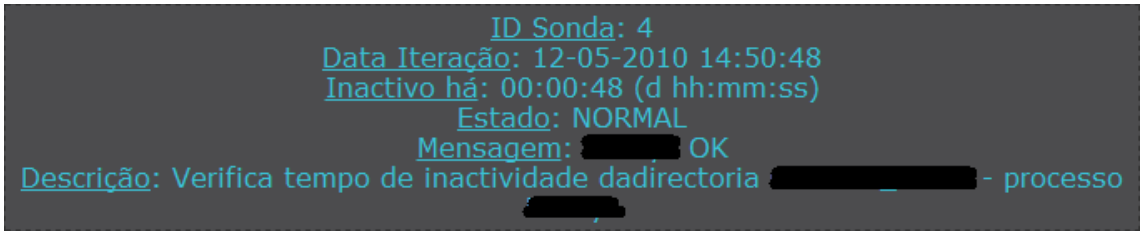


Figura 3.6: Sputnik - Descrição sonda beta

No âmbito do meu PEI actualizei as descrições das sondas, de maneira a conterem informação mais detalhada sobre a monitorização (figura 3.7).

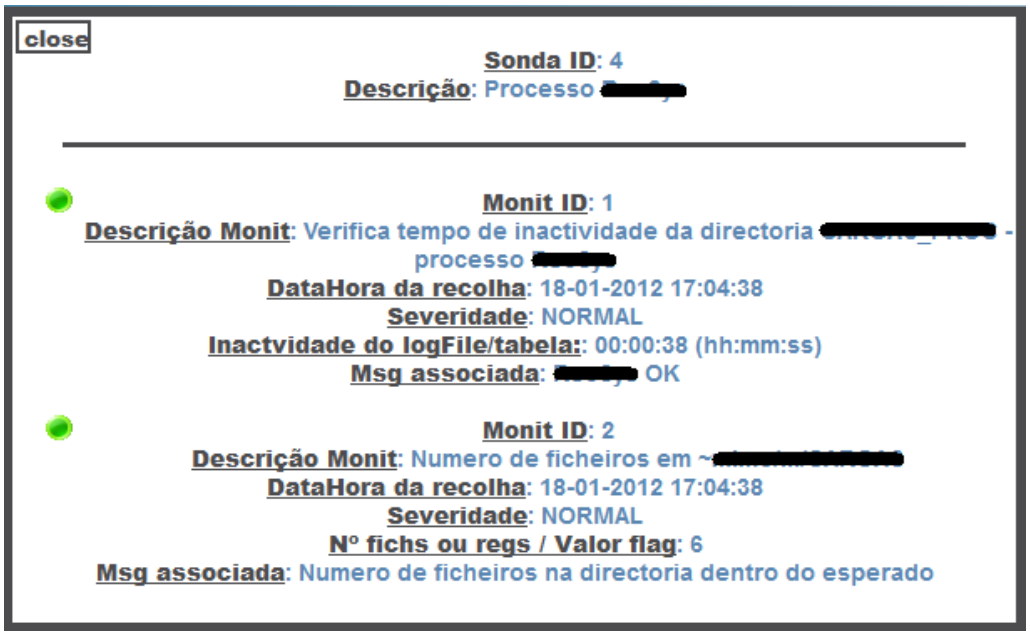


Figura 3.7: Sputnik - Nova descrição sonda

Posteriormente comecei a desenvolver a aplicação *Sputnik Project* que, para além da representação gráfica dos circuitos de sondas, contém também a nova consola de gestão para clientes, o novo processo de *Rollout*, consulta de *logs* das monitorizações e ainda *links* úteis para as equipas da OA.


Figura 3.8: Sputnik Project - Página Inicial

No menu ”Gestão”é possível criar/editar/listar *dashboards*, sondas, monitorizações, *thresholds* e variáveis de sistema. Uma particularidade nesta versão é a possibilidade de se fazer *upload* da imagem de fundo do *dashboard*, para o servidor. A imagem serve para disponibilizar as sondas com as respectivas monitorizações.

O menu *Rollout* dá acesso à página do novo processo de *Rollout*, onde o utilizador pode definir os locais onde pretende que as sondas apareçam.

O menu *Dashboards* permite escolher um *dashboard* colocando-o numa página *stand alone*, por exemplo para ser mostrado num monitor de modo a que as equipas de operação possam controlar os circuitos.

O menu *Logs* permite consultar os *logs* das iterações das monitorizações ao longo dos dias, contém vários campos de pesquisa, desde o nome do *dashboard* até à severidade das iterações.

Desta forma é possível representar fluxos aplicacionais, *dashboards*, e outras realidades que se adaptem às características da aplicação.

### 3.4.1 Esquema de Cores das Sondas

Existem quatro cores definidas para representar os possíveis estados de cada sonda. Cada um desses estados é representado por um led de diferente coloração. As convenções de cores são as seguintes:

Verde - Indica que o estado da monitorização efectuada pela sonda está OK;

Amarelo - Indica que existe ligeiro atraso, acumulação ou fluxo de dados intenso na monitorização;

Vermelho - Indica indisponibilidade, atraso significativo ou fluxo parado na monitorização;

Cinzento - Indica que não foi possível colectar o estado da sonda. A sonda está inactiva;

## 3.5 Consola de Gestão (para clientes)

A consola de gestão permite ao cliente gerir os seus alarmes, podendo adicionar, editar e remover *thresholds*, bem como alterar mensagens dos alarmes. O processo que existia, era extremamente arcaico e demorado de executar.

Este processo já era efectuado via aplicação *Web*, mas existia apenas um caminho para chegar às sondas, monitorizações, ou *thresholds* e muita da informação sobre os mesmos tinha de ser colocada manualmente no *html* das páginas.

A nova solução para evolução desta ferramenta que desenvolvi disponibiliza uma zona de gestão em que o cliente pode adicionar, editar ou remover as suas monitorizações à sua instância, de forma mais intuitiva e sem passar por tantos menus, ficando as alterações efectivadas no momento. Na edição das monitorizações será possível alterar, por exemplo, *thresholds* ou mensagens dos alarmes. A divisão das sondas/monitorizações será feita por *dashboard*, diminuindo a listagem de monitorizações mostrada ao utilizador e facilitando encontrar o pretendido.

A navegação em todas as secções da aplicação são na forma de *popups*, havendo quatro níveis (no máximo) de profundidade, de forma a que o utilizador ”sinta”que tem sempre o controlo total da aplicação.

Nas imagens seguintes é mostrado o fluxo para a criação de um *dashboard* até ao *threshold* de uma monitorização.


Figura 3.9: Sputnik - Menu *Gestao*

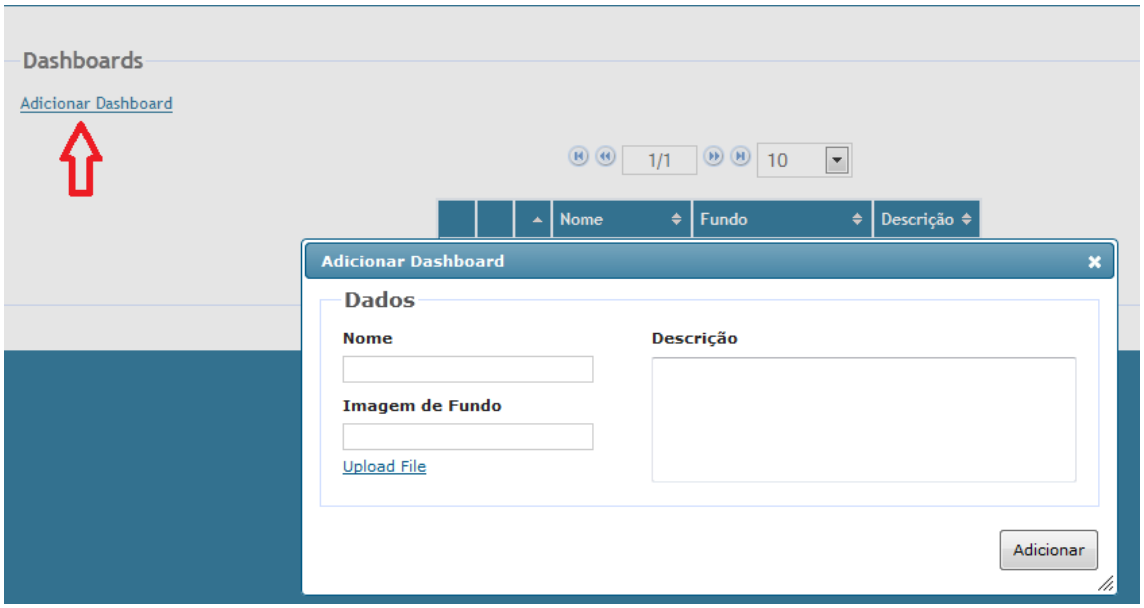


Figura 3.10: Sputnik - Adicionar *dashboard*

Na figura 3.10 são mostrados os campos para adicionar um *dashboard*, esta representação serve para que a associação entre ”imagem”e sonda se torne mais intuitiva e desta forma também é possível associar a imagem de fundo onde devem aparecer as sondas.

Na figura 3.11 pode ver-se que através de um *link* na listagem dos *dashboards* é possível aceder à listagem das sondas desse *dashboard*, bem como adicionar novas sondas.

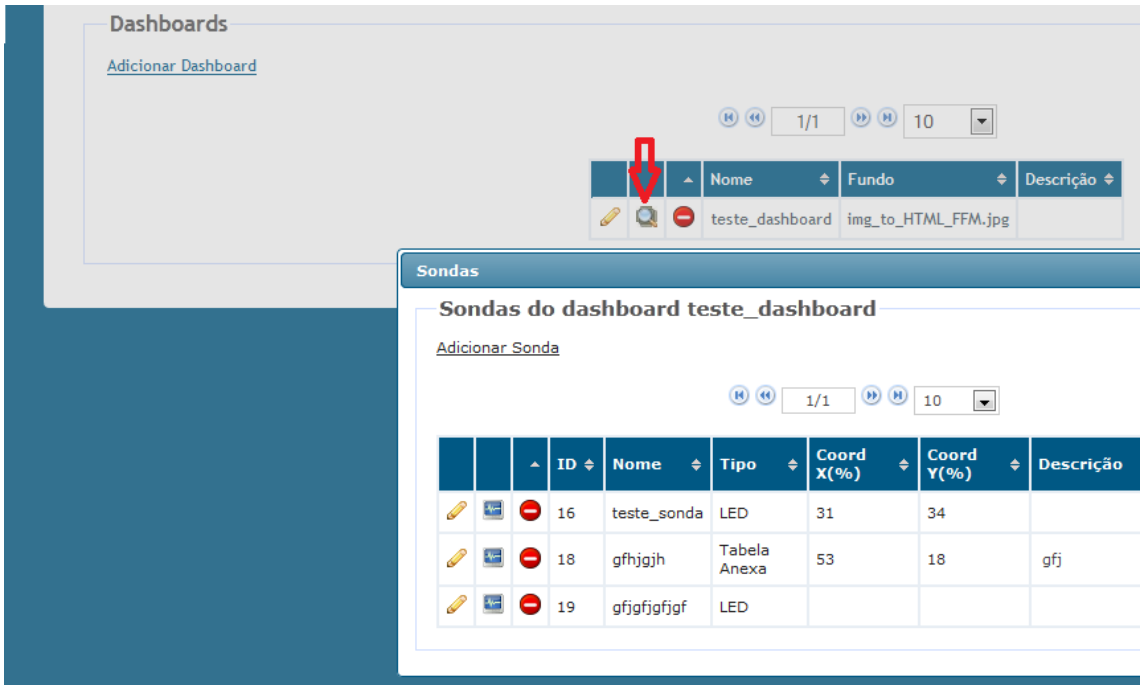


Figura 3.11: Sputnik - Adicionar sonda

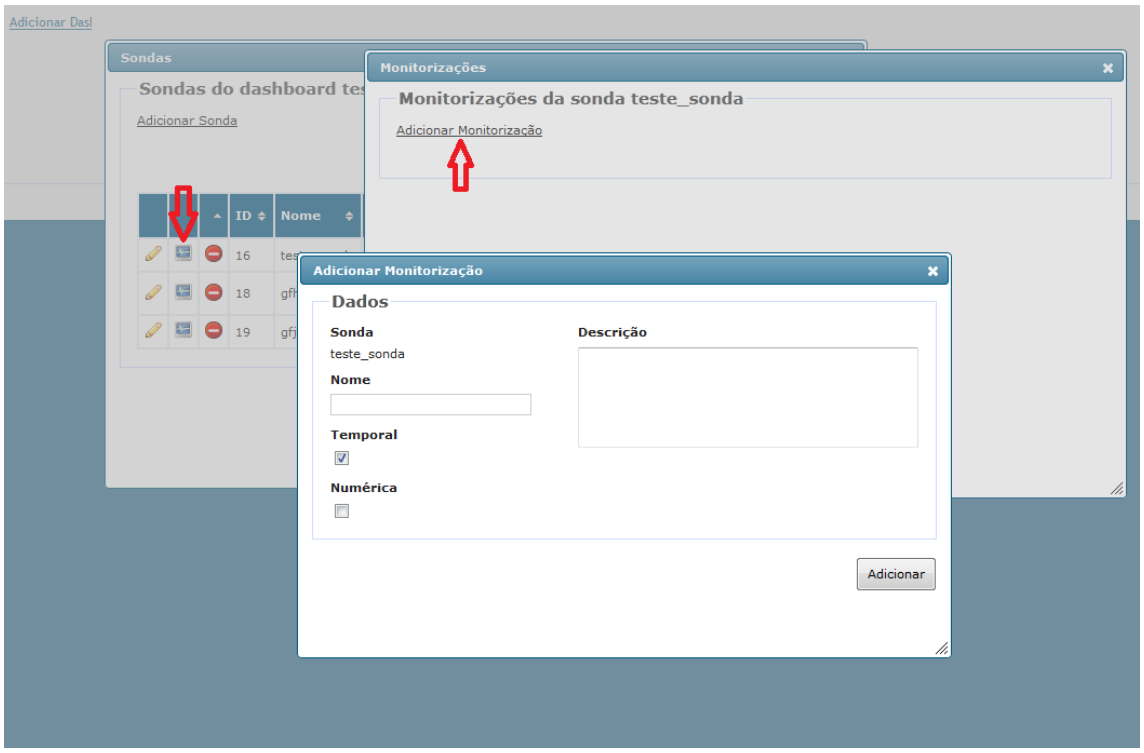


Figura 3.12: Sputnik - Adicionar monitorização

Tal como ao adicionar sondas aos *dashboards*, para adicionar monitorizações às sondas, também existe um *link* (figura 3.12), na listagem das sondas que permite essa acção.

Para adicionar *thresholds* a monitorizações procede-se da mesma forma, como se pode

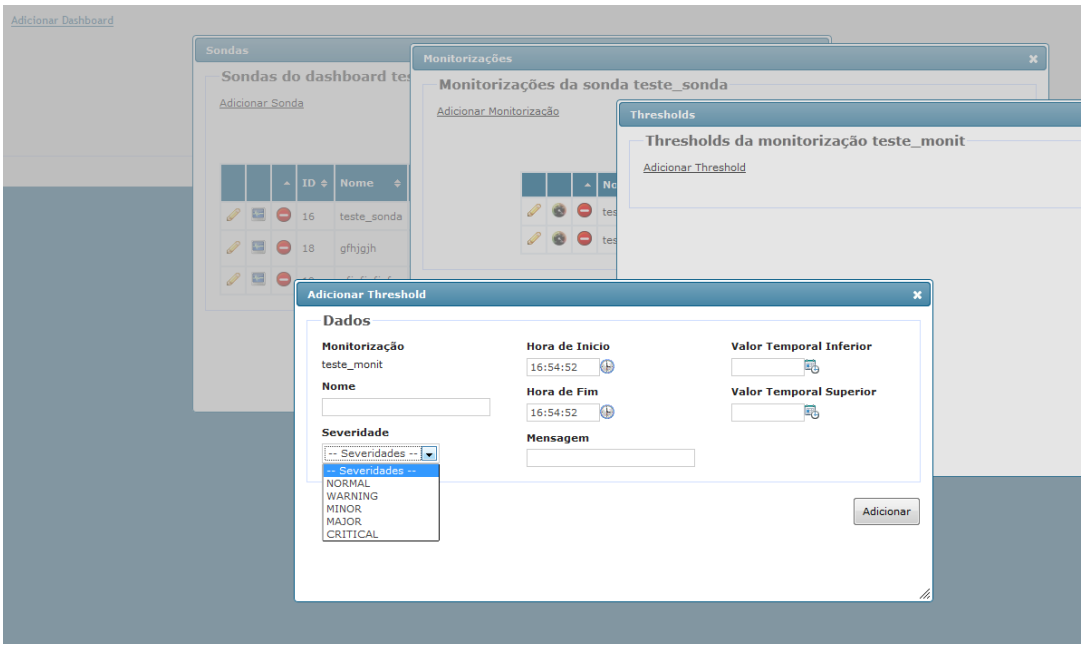


Figura 3.13: Sputnik - Adicionar *threshold*

ver na figura 3.13.

Até aqui foi mostrada uma maneira de navegação para adicionar sondas, monitorizações e *thresholds* mas, tal como, para adicionar um *dashboard* o utilizador acede através do menu de ”Gestão”, também é possível criar o resto acedendo através do menu. Na imagem 3.14 é mostrado como se adiciona um *threshold* através do submenu *Thresholds*. Para adicionar sondas ou monitorizações é equivalente escolhendo, respectivamente um *dashboard* ou um *dashboard* e uma sonda.

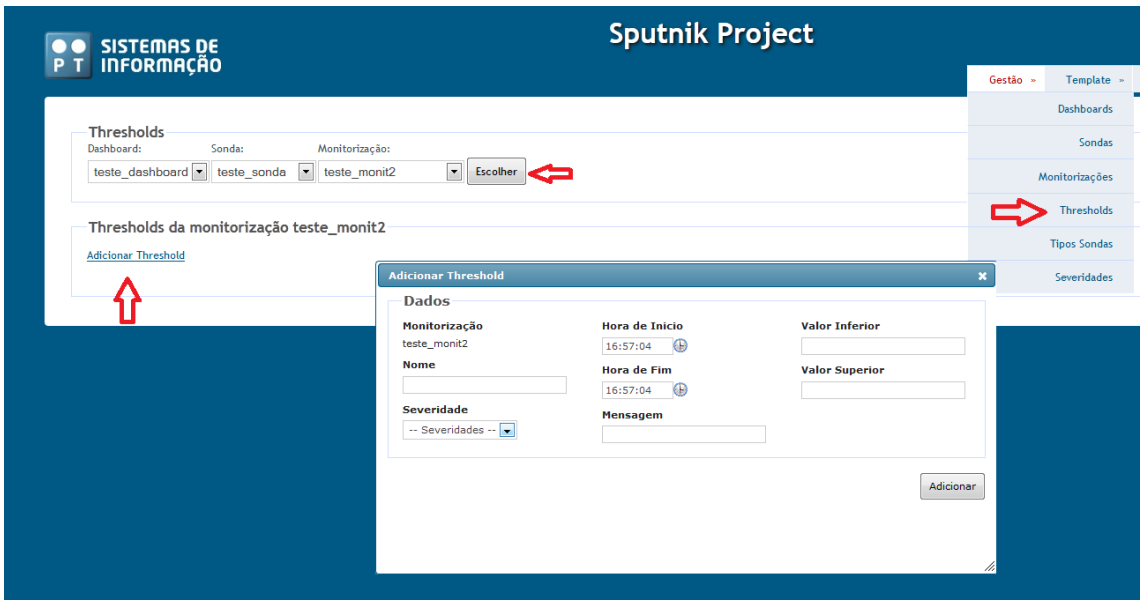


Figura 3.14: Sputnik - Adicionar *threshold* (submenu)

Como se pode verificar nas duas últimas imagens, a navegação entre as secções da ”Gestão”pode ser efectuada de duas formas distintas, através dos *popups* ou através dos submenus.

Esta ferramenta não só simplifica muito a instanciação de novos interfaces *Sputnik*, como retira esforço manual da equipa de Monitorização Aplicacional até aqui responsável pelo processo. Mais robustez, menos margem de erro e menos esforço.

## 3.6 Carregamento Massivo de Dados

Normalmente, um *dashboard* contém várias monitorizações e a tarefa de adicionar uma a uma via *Web* não é de todo eficaz. Focado neste pensamento, decidi criar um ficheiro *Excel* para servir de template, onde os utilizadores têm uma *sheet* dedicada a cada um dos elementos necessários para a criação de um novo circuito.

O ficheiro *Excel* é criado via *C#* através da biblioteca *Microsoft.Office.Interop.Excel*. Para além das várias *sheets* com cabeçalhos para o utilizador saber o que está a preencher, também criei macros em *VBS*, que são inseridas no ficheiro, também, via *C#*, para quando um utilizador está a preencher, por exemplo, os campos de uma sonda, poder escolher o *dashboard* ao qual pertence a sonda. Esta escolha é feita através de uma *dropdown-list* que contém os nomes dos *dashboards* preenchidos na primeira *sheet*. Isto acontece também nas *sheets* das monitorizações (escolhendo a sonda) e *thresholds* (escolhendo a monitorização).

Utilizando a mesma biblioteca, também criei os métodos para ler um template preenchido e inserir os valores na BD. Quando estive a migrar o circuito das recargas da TMN, entre outros, para o novo *Sputnik*, carreguei todos os dados através do template, poupando assim várias horas de trabalho.

## 3.7 Automatização do Rollout

O processo de *Rollout* é um processo para criação de novas instâncias *Sputnik*, isto é, novas plataformas de alarmística para novos domínios. Por exemplo, garantir a monitorização do fluxo de aprovisionamento de cartões e serviços no negócio móvel: activação de *MMSs*, activação de *Roaming*, alteração de tarifários, etc. Antes da nova versão, este processo era bastante arcaico, sendo tudo feito à mão.

A automatização deste processo consistiu na criação de um menu na aplicação Web (figura 3.8) onde os clientes podem, facilmente, adicionar as sondas aos interfaces gráficos para disponibilizar as monitorizações ou estatísticas das alarmísticas nos *dashboards*. A posição de cada sonda, que actualmente tem de ser definida no *web.config*, passou a ser dinâmica, podendo o cliente definir directamente na aplicação e ter um pequeno *preview* do *layout* que está a construir.

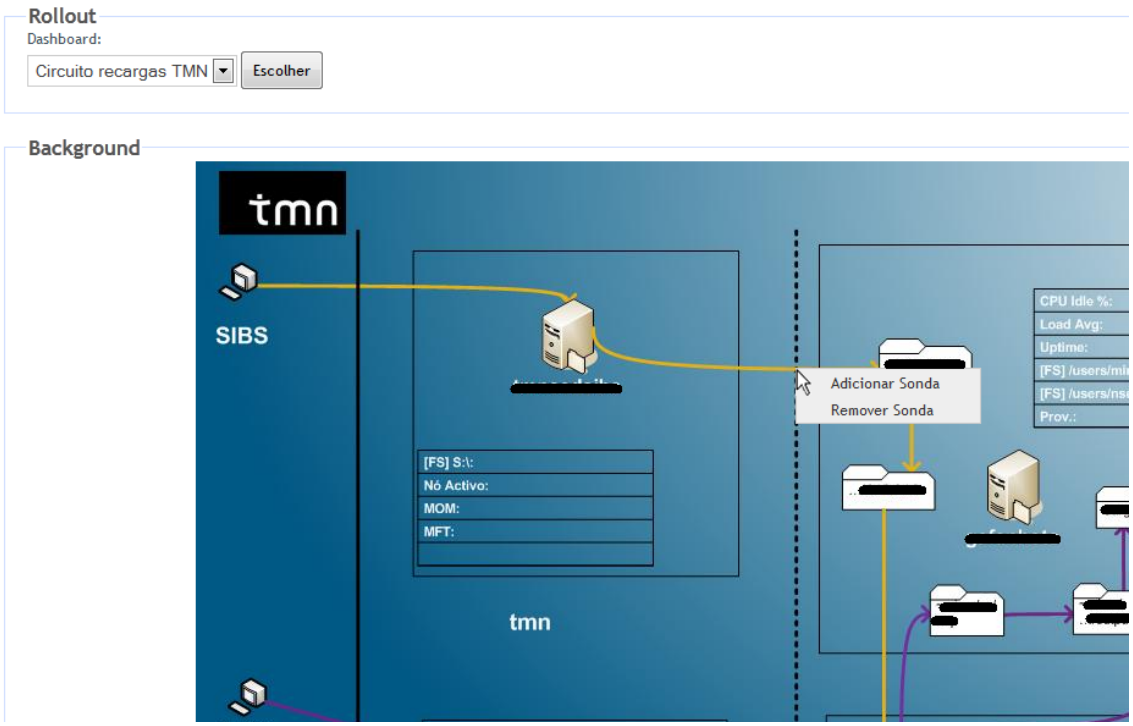


Figura 3.15: Sputnik (*Rollout*) - Escolher acção

Como se pode ver na imagem 3.15, ao aceder ao menu *Rollout* é disponibilizada uma *dropdownlist* com os nomes dos *dashboards* que contêm imagem de fundo. Após escolher o *dashboard* pretendido, é disponibilizada a sua imagem de fundo, onde o utilizador pode seleccionar a posição onde quer que a sonda apareça. Para isso basta carregar no botão direito do rato em cima da imagem e aparece um menu onde se pode escolher adicionar ou remover sondas.

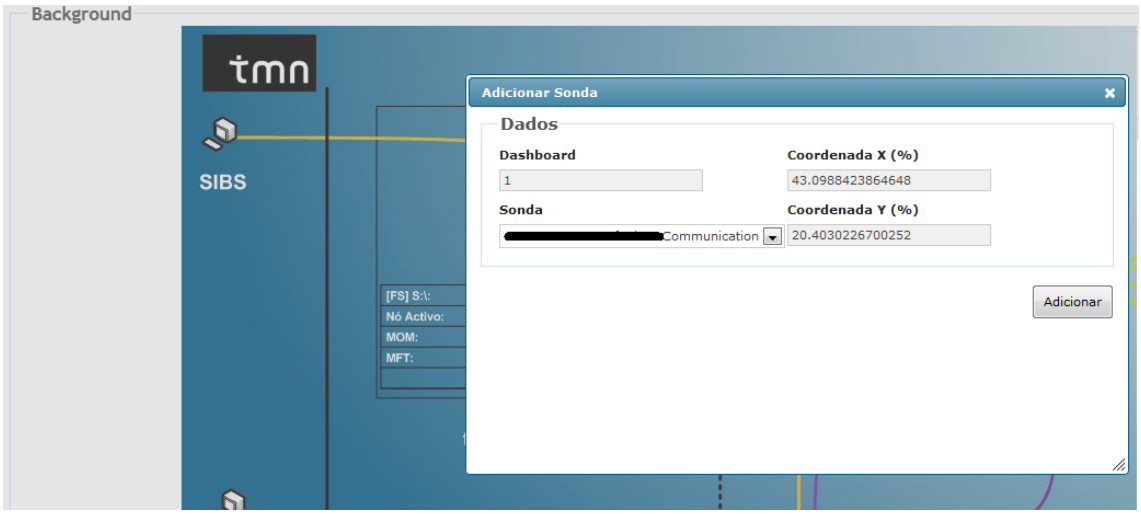


Figura 3.16: Sputnik (*Rollout*) - Adicionar sonda

Ao escolher ”Adicionar Sonda”é disponibilizado um *popup* (figura 3.16) com o nome

do *dashboard*, as coordenadas da imagem onde a sonda vai aparecer (em percentagem) e uma lista para escolher a sonda a colocar nessa posição. (Nota: tanto o *dashboard*, como as sondas já foram inseridas, anteriormente, na BD). Após adicionar a sonda a imagem é actualizada mostrando onde a sonda foi colocada (figura 3.17).

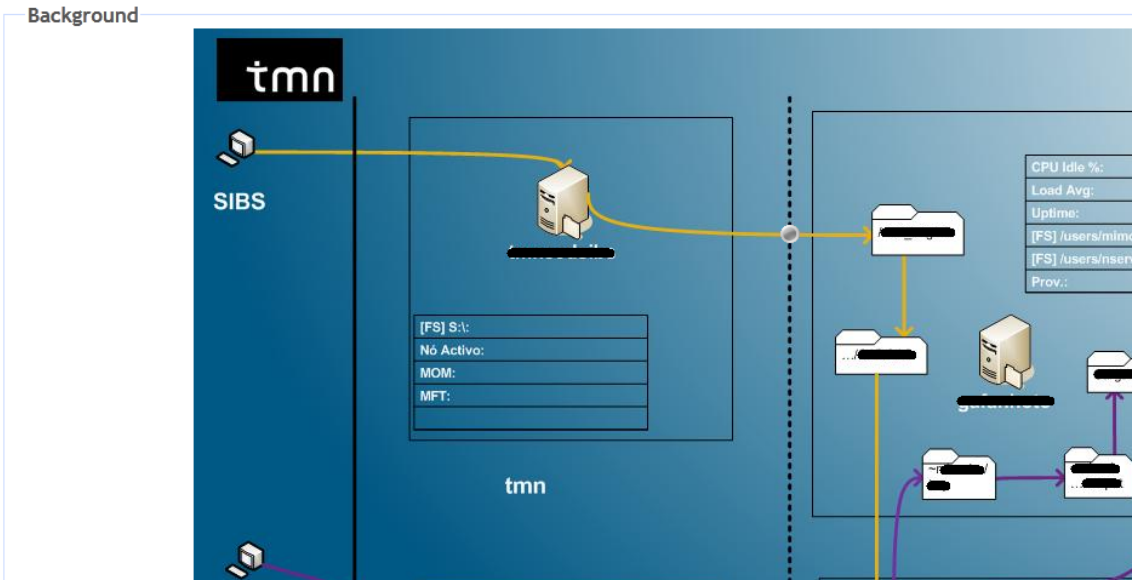


Figura 3.17: Sputnik (*Rollout*) - Resultado da colocação da sonda

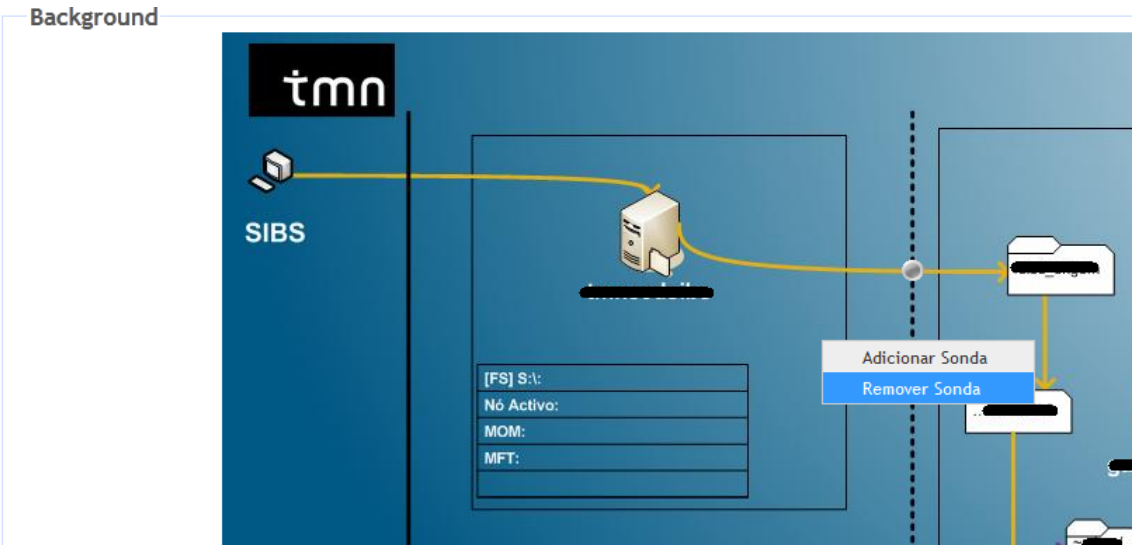


Figura 3.18: Sputnik (*Rollout*) - Remover sonda

A opção ”Remover Sonda”pode ser acedida em qualquer parte da imagem (figura 3.18) pois a sonda é removida escolhendo o nome e não pelas coordenadas.

Tal como para adicionar a sonda, para remover também aparece um *popup* (figura 3.19) para escolher a sonda a apagar. Após carregar em remover a sonda é apagada da imagem.

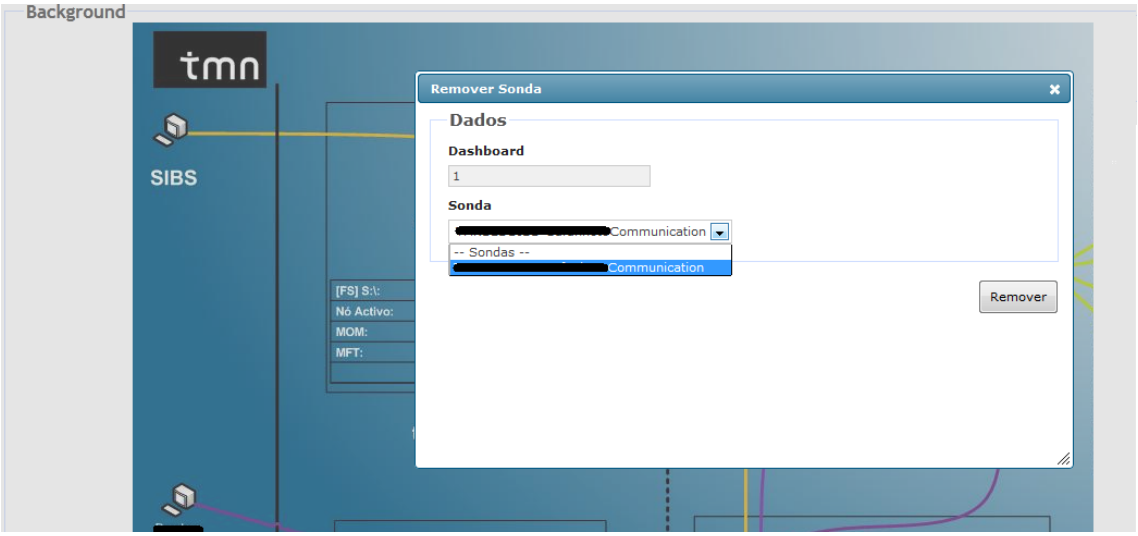


Figura 3.19: Sputnik (*Rollout*) - Remover sonda (*popup*)

## 3.8 Análise de Plataforma de Agentes

Conforme é mostrado nos *milestones* (secção 2.2) do Planeamento, inicialmente foi pensado utilizar-se uma plataforma de agentes independente de SO. A minha primeira ideia foi utilizar o *Java*, devido à sua independência em relação ao SO em que está a correr e devido à minha experiência académica nesta linguagem.

Após análise de algumas plataformas, como *Jack* [17] e *RETSINA* [24], optei por utilizar a *framework JADE* (*Java Agent DEvelopment Framework*) [23] que é um ambiente para desenvolvimento de aplicações baseada em agentes conforme as especificações da FIPA (*Foundation for Intelligent Physical Agents*) para interoperabilidade entre sistemas multi-agentes totalmente implementado em Java. O principal objectivo do *Jade* é simplificar e facilitar o desenvolvimento de sistemas multi-agentes garantindo um padrão de interoperabilidade entre os sistemas através de um abrangente conjunto de agentes de serviços de sistema, os quais tanto facilitam como possibilitam a comunicação entre agentes.

O *Jade* oferece várias funcionalidades para a programação de agentes, entre as quais:

- Plataforma distribuída de agentes para ligações entre máquinas, via *RMI* (Remote Method Invocation);
- Ferramentas de *Debugging*;
- Suporte a execução de actividades de agentes múltiplas, paralelas e concorrentes;

No entanto, os problemas começaram logo à partida quando pedi autorização para instalar o *Jade* nas máquinas que monitorizamos e descobri que muitas nem tinham uma *JVM* (*Java Virtual Machine*) instalada e existiram logo vários entraves à sua instalação

devido à quantidade de processos que as máquinas já executavam e aos poucos recursos que sobravam.

Esta situação tornou-se num factor decisivo para deixar de parte a criação de um Agente *Java* e optei por uma solução mais simples, leve a nível de recursos da máquina e que já estava em desenvolvimento pela minha equipa (Monitorização Aplicacional), tendo a maioria das características que procurava para o *Sputnik*, tais como:

- processo automático de envio de scripts para as máquinas a monitorizar;
- concepção de *scripts* e monitorizações tanto para sistemas *Unix* e *Windows*;
- *scheduling* próprio;

Esta integração é possível através da inclusão da chamada do *Spunik WS*, nos *scripts* das monitorizações partilhadas entre *Mon-IT* e *Sputnik*. Assim estes *scripts* passaram a chamar as funções disponibilizadas pelo *Sputnik WS*. Esta simplicidade de integração foi outro factor decisivo para não usar o Agente Java.

A *Mon-IT* (secção ) é uma aplicação de monitorização que utiliza um *script wrapper* para executar os vários *scripts* responsáveis pelas monitorizações. Existe uma versão para sistemas *Windows* e outra para sistemas *Unix*. Basta colocar o *script wrapper* no *crontab* da máquina (sistema *Unix*) ou como *schedule task* (sistema *Windows*) para ser executado de minuto a minuto. Os *scripts* que são executados através do *wrapper* não necessitam ser controlados pelo SO pois o *wrapper* contém um *scheduling* próprio e gerido pelo *Mon-IT WS* que despoleta os alarmes, quando necessários.

Todas as configurações são efectuadas via *Web* e as alterações são enviadas para a máquina respectiva através de um processo automático. Este processo consulta a BD em intervalos de X minutos (configurável via *Web*) e caso existam novas monitorizações, liga-se à respectiva máquina via SSH (*Secure Shell*) enviando:

- o *wrapper* (caso não exista ou tenha sido actualizado);
- o script que colecta os valores a monitorizar (caso não exista ou tenha sido actualizado);
- o ficheiro de *scheduling* do *wrapper* (com as execuções para dois dias a contar da data actual);

Em cada execução do *wrapper* são lançados os vários scripts, sequencialmente, que tenham um *timestamp* inferior ou igual à data actual. Cada valor que o *wrapper* recebe, envia-o para o *webservice*. O *webservice* verifica a que monitorização o valor pertence e consulta os seus *thresholds*, validando se é necessário despoletar alarme.

## 3.9 Integração de Aplicações com o Web Service

O Webservice do *Sputnik* para além de funcionar como o motor da aplicação3.3 também foi construído de maneira a poder comunicar com outras plataformas, pois pretende ser uma ferramenta de interligação da plataforma com outras aplicações e ferramentas, como o *HP OpenView* para enviar alarmes ou a *Gateway Asterisk* para despoletar chamadas automáticas para os responsáveis pelo alarme.

Mas a integração fulcral é com a *Checklist*, pois torna-se possível saber a relação que um alarme tem com os restantes *CIs*. Isto é importante para saber a afectação que existe no serviço, ao nível do negócio.

A ligação entre o *Sputnik* e a *Mon-IT*, conforme já foi dito, é feita através da chamada do *Sputnik WS* nos *scripts* partilhados entre as duas aplicações.

Através da integração com a aplicação *Mon-IT* 3.8 foi possível dinamizar as monitorizações, pois processos da *Mon-IT* podem ser representados no *Sputnik* sem ser necessário haver outro processo semelhante a chamar apenas o *Sputnik WS* e com *scheduling* e execução controladas pelo SO.

Este *webservice* foi desenvolvido em *C#* e os pedidos e respostas podem ser recebidos/enviados em formato *XML* ou *string*, de maneira a facilitar a integração com aplicações mais ou menos recentes.

## 3.10 Processo de *Housekeeping*

O processo de *Housekeeping* consiste na limpeza da tabela *Sput_Log*. Esta tabela tem tendência a aumentar astronomicamente pois para cada monitorização é guardado em *log* o valor de cada iteração (normalmente de 5 em 5 minutos).

Este corre todos os dias à 01:00 e cria um ficheiro *CSV* (*Commaseparated values*) com os *logs* das monitorizações do dia anterior. No primeiro dia de cada mês, os ficheiros *CSV* do mês anterior são comprimidos num único ficheiro em formato *zip* e os ficheiros *CSV* são apagados. Apesar de neste momento o processo estar a criar ficheiros com *logs* do dia anterior, o número de dias que os *logs* devem ficar guardados na tabela é definido através de uma *sysvar* e o seu valor pode ser alterado via *Web*.

Para criar os ficheiros *CSV* utilizo apenas as bibliotecas do *C#*, para cada *log* crio uma linha com os valores de cada coluna divididos por ';' e guardo num objecto *StringWriter*, no final utilizo o método estático *WriteAllText(file, text)* da biblioteca *File* para escrever o texto para o ficheiro *CSV*.

Para criar o ficheiro *zip* utilizo uma biblioteca *opensource* chamada *ZipSharp* que contém métodos para comprimir ficheiros nos formatos *bzip2*, *gzip*, *tar* e *zip*. Através do método nativo do *C#*, *Directory.GetFiles(path, file_name)* obtenho os nomes dos ficheiros do mês anterior e guardando-os numa lista. Esta lista é utilizada para criar objectos

*ZipEntry* e adicionados a um objecto *ZipOutputStream* que cria o ficheiro *zip*. Tanto o objecto *ZipEntry* como o *ZipOutputStream* vêm da biblioteca *ZipSharp*.

## 3.11 Documentação

Como documentação, criei três manuais, um de criação/gestão de *dashboards*, sondas, monitorizações e *thresholds*, uma para o processo de *Rollout* e outro para comunicação com o *Webservice*. Criei, também, uma apresentação em *PowerPoint* para ser utilizada no ”Dia da OA”, que consistiu em apresentarmos aos directores das outras áreas da empresa, as aplicações que desenvolvemos dentro da nossa área, com o intuito não só de as vender mas também de mostrar o nosso trabalho e ter a possibilidade de conseguir novos projectos.

# Capítulo 4

# Gateway Asterisk

## 4.1 Contextualização

O *Asterisk* é um *software* que transforma um computador comum num servidor de comunicações, implementando um telefone *PBX* (*Private Branch Exchange*). Foi criado em 1999 por *Mark Spencer* da *Digium*. Tal como qualquer *PBX*, permite que telefones ligados ao *PBX*, efectuem chamadas entre si e consigam ligar-se a outros serviços telefónicos, como *PSTNs* (*Public Switched Telephone Network*) e *VoIP* (*Voice over Internet Protocol*), podendo ligar-se também a equipamentos *GSM* (*Global System for Mobile Communications*), caso do meu PEI. É usado por pequenas empresas, grandes empresas, os *Call Centers*, operadoras e governos em todo o mundo. O *Asterisk* é livre e de código aberto.

A equipa da Produção trabalha de forma contínua (24x7) e recebe diariamente dezenas de alarmes, muitos deles indicando apenas, nas instruções, para contactar a prevenção da equipa responsável pelo alarme (a prevenção de uma equipa consiste num elemento da mesma estar contactável 24x7 para entrar em acção sempre que existam problemas). O tempo gasto pelos recursos da equipa, por vezes, era enorme pois muitas vezes só à terceira ou quarta tentativa é que a prevenção atendia. Esta situação multiplicada por dezenas de alarmes torna-se insustentável e passível de ser melhorada e optimizada.

No contexto do meu PEI, o piloto *Gateway Asterisk* foi criado para libertar a Produção do *report* de alarmes *HP-OVO* que implicam só um contacto à equipa de prevenção, automatizando assim totalmente este tipo de alarmística.

O construção da plataforma consistiu na configuração da aplicação *Asterisk Now* numa máquina virtual com SO *Linux*, configuração do modem GSM, na criação de duas aplicações em *.Net*, uma aplicação de consola e outra *Web* e ainda um *Webservice* para integração com outras aplicações. Neste momento o piloto já se encontra ultrapassado e a aplicação já se encontra a reportar vários alarmes recebidos pela Produção OA.

Esta aplicação envolve tecnologias, como bases de dados *SQL Server*, aplicações *Web* em *ASP.NET MVC 3*, *webservices* em *C#* recebendo e respondendo a pedidos em *XML*,

*LINQ*, *shellscript*, entre outras.

## 4.2 Configurações

### 4.2.1 Asterisk Now

A aplicação utilizada foi o *Asterisk Now* que é uma aplicação *open-source* para *Linux*. Para instalar a aplicação, foi criada uma *VM* (*Virtual Machine*) com o SO Linux. Foram, também, criados três *scripts* para criar os ficheiros de chamadas (*.call*) e configurada a aplicação *Asterisk Now* e o *Modem GSM*.

Esta aplicação, depois de configurada, encarrega-se de enviar automaticamente e de forma instantânea, para o *Modem GSM*, qualquer ficheiro de chamada que se encontre no directório da própria aplicação *Asterisk*, chamado *outgoing*.

#### 4.2.1.1 Criação de Scripts

Para criar os ficheiros das chamadas e montar o sistema pretendido, desenvolvi três *scripts* em *shellscript* para auxiliarem no processo.

O *trans2asterisk.sh* encarrega-se de receber os ficheiros, com os contactos para as chamadas, vindos da aplicação *Windows* (secção 4.4), executar o processo (*MakeCall.sh*) que cria os ficheiros das chamadas e de enviar as respostas de sucesso ou insucesso de volta.

O *script MakeCall.sh* cria um ficheiro *.call* que contém, para além do número de destino, várias opções para o *modem GSM* executar, como o canal (neste caso é utilizado o *SIP - Session Initiation Protocol*), tempo de espera, máximo de tentativas, tempo entre tentativas, contexto (configurado no *GUI* (*Graphical User Interface*) do *Asterisk Now*) e extensão (estas opções encontram-se explicadas na secção 4.2.1.2). O ficheiro depois de criado é movido para o directório *outgoing*.

No final da execução do *script MakeCall.sh*, é executado o terceiro *script*, *CheckCall.sh*, que consulta o *log* de sistema do canal *SIP* até a chamada terminar e valida se a mesma foi atendida e efectuado o *acknowledge* da mesma. O *acknowledge* da chamada consiste no utilizador inserir um conjunto de caracteres definido no contexto, caso contrário volta a tentar passados X minutos até um número máximo de Y tentativas (X e Y configurados via *Web*).

#### 4.2.1.2 Via Interface Gráfico

Em primeiro lugar vou apresentar três conceitos, extremamente importantes, utilizados pelo *Asterisk*:

- Extensão - utilizador com capacidade de se ligar e usar o sistema para efectuar ou receber chamadas, usar *voicemail* e outras funcionalidades do *Asterisk*

- Contexto - conjunto de extensões que indica, dentro do *dialplan*, ao *Asterisk* onde colocar as chamadas recebidas para cada extensão.

- *Dialplan* - conjunto de contextos, ou seja, o plano de ligações possíveis para o servidor que está a ser configurado.

Nas configurações efectuadas via *GUI*, criei quatro contextos (figura 4.1), no ficheiro *extensions.conf*. Apenas o primeiro (*ivr*) é utilizado na criação do ficheiro da chamada (ver secção 4.2.1.1), os outro três contextos, *ivrSuccess*, *ivrRepeat* e *ivrFail* são utilizados apenas como funções no contexto *ivr*.

Como se pode ver na figura 4.1, no *ivr* criei uma extensão com várias acções a efectuar no caso da chamada ser atendida, primeiro cria umas *flags* para o *script trans2asterisk.sh* saber que está uma chamada em execução. Depois vem a leitura do código inserido pelo destinatário em caso de inserir o código correcto é passada uma mensagem de confirmação e a chamada termina (*ivrScuccess*), caso contrário espera novamente pela inserção de um código válido (*ivrRepeat*), à terceira tentativa errada a chamada falha e é passada uma mensagem de despedida (*ivrFail*). Todos os ficheiros áudio encontram-se numa pasta de sistema do *Asterisk*. Configurei ainda um contexto já existente, o *voibridge* onde coloquei o canal a ser usado pela extensão que criei.

Depois de configurar a extensão e contextos, foi necessário configurar a aplicação para chegar ao *Modem GSM* (figura 4.2). Esta configuração é feita no ficheiro de configuração do canal *sip.conf*. Neste ficheiro criei um novo contexto com o *host* e *port* do *modem*, o domínio de partida, ou seja, o *host* onde está o *Asterisk*, entre outras opções do *Asterisk*.

### 4.2.2 Modem GSM (Topex)

#### 4.2.2.1 Via Interface Gráfico

Na configuração do modem, acedi à página de administração e configurei, em primeiro lugar, o cartão móvel utilizado nas chamadas, desta maneira ao modem consegue acesso à rede da TMN (figura 4.3). O código *26806-TMN*, no campo *Operator* é o código internacional da TMN.

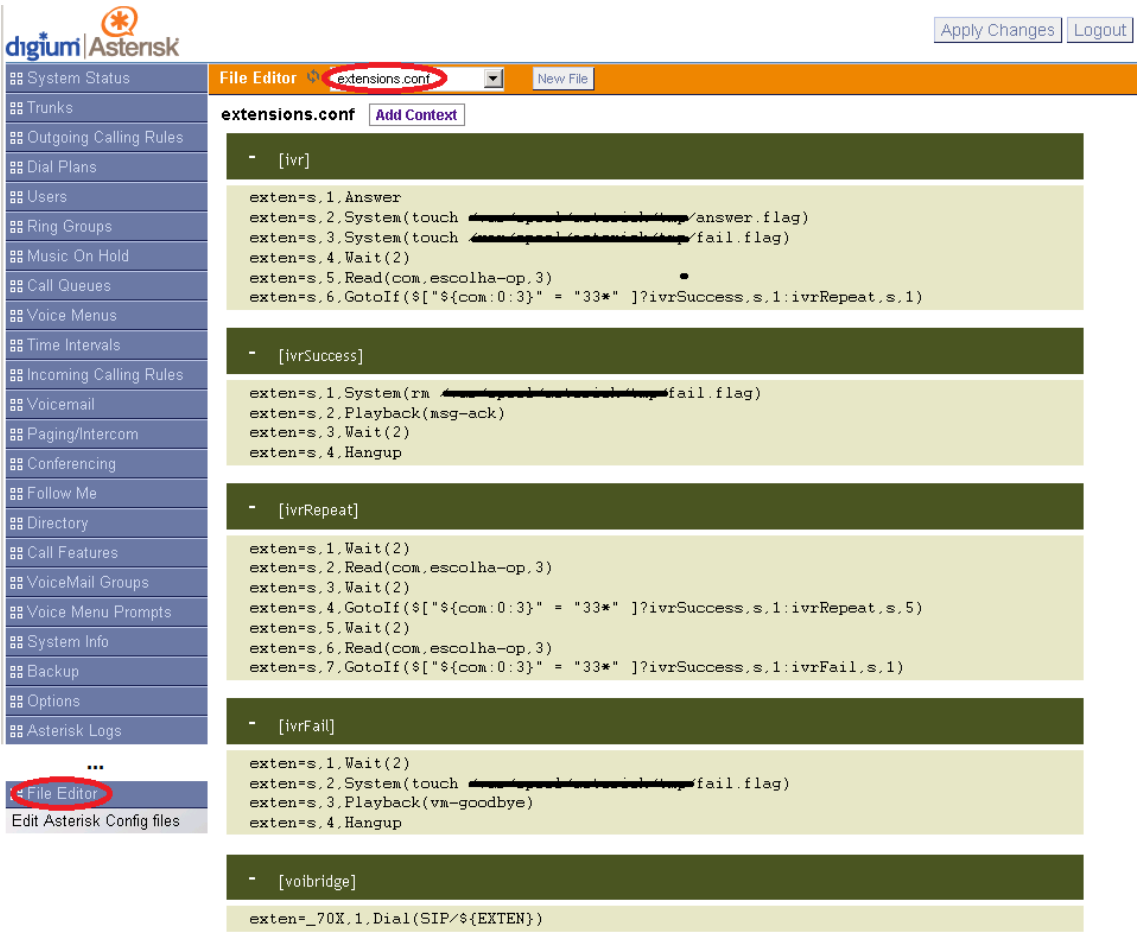


Figura 4.1: Asterisk Now (*GUI*) - Adicionar Extensão)

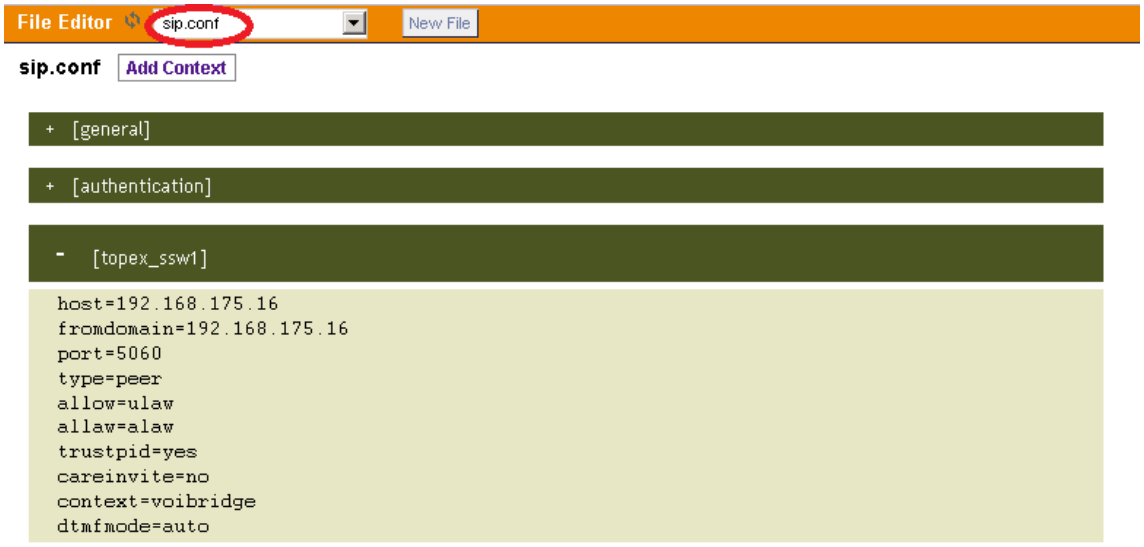


Figura 4.2: Asterisk Now (*GUI*) - Associar Modem)

No menu *PBX -> Settings* (figura 4.4) adicionei o IP do servidor do *Asterisk*, o protocolo utilizado (*SIP*) e o porto de comunicação entre o modem e o servidor. Esta ligação

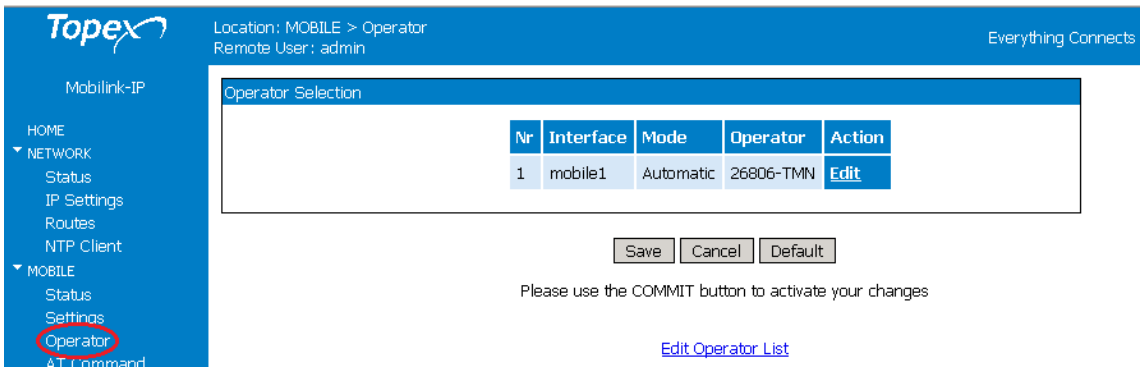


Figura 4.3: Topex - Configuração da rede

serve para o *Asterisk* conseguir enviar os ficheiros das chamadas para o modem.

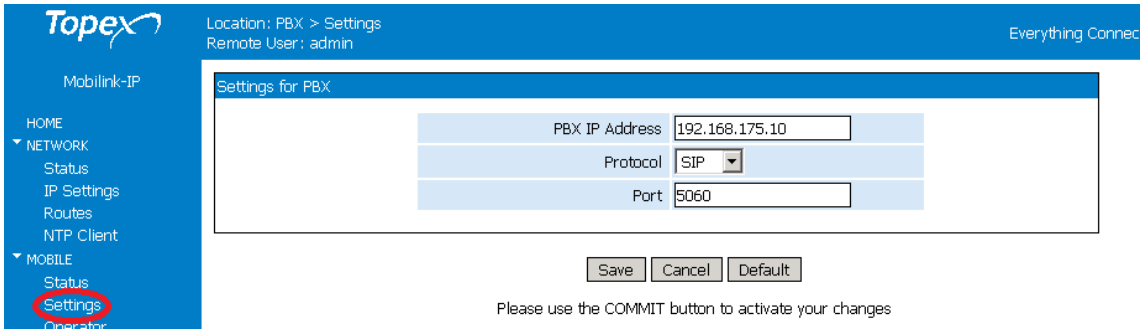


Figura 4.4: Topex - Configuração do acesso ao PBX

No final é só adicionar as opções das chamadas (figura 4.5), como a força do sinal no envio e recepção, o *CLIR* (*Calling Line Identification Restriction*) ou o *CLIP* (*Calling Line Identification Presentation*). O *CLIR* indica se o destinatário vê o número de emissor ou se vê uma *label* a indicar que o número é privado e o *CLIP* indica se a *label* é mostrada ou não. No caso de estarem os dois campos a '*No*', o destinatário recebe uma chamada sem qualquer identificação.

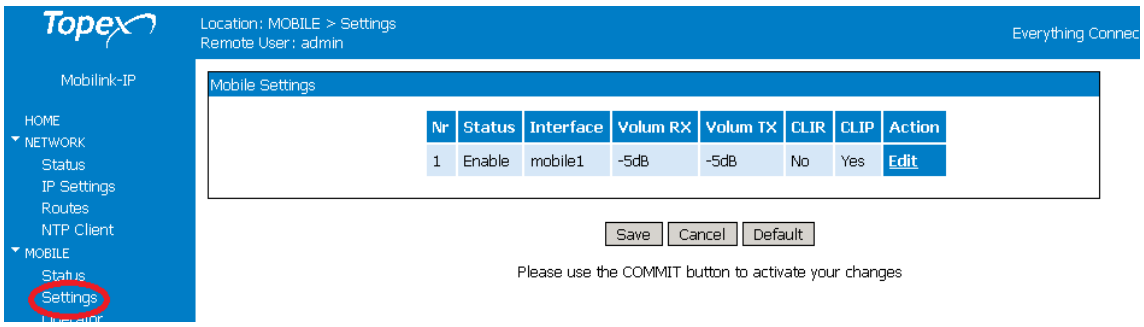


Figura 4.5: Topex - Configuração do cartão móvel

A partir deste momento o *Asterisk* faz a gestão e a utilização deste recurso (funções de *PBX*).

## 4.3 Modelo de Dados

O modelo de dados (figura 4.6) foi criado/focado na integração com o *HP-OVO*, pois é a ferramenta oficial da PT para envio de alarmes. Devido a este facto, as tabelas que contém os alarmes mapeados e os alarmes recebidos têm colunas equivalentes às opções que vêm nos alarmes do *HP-OVO*.

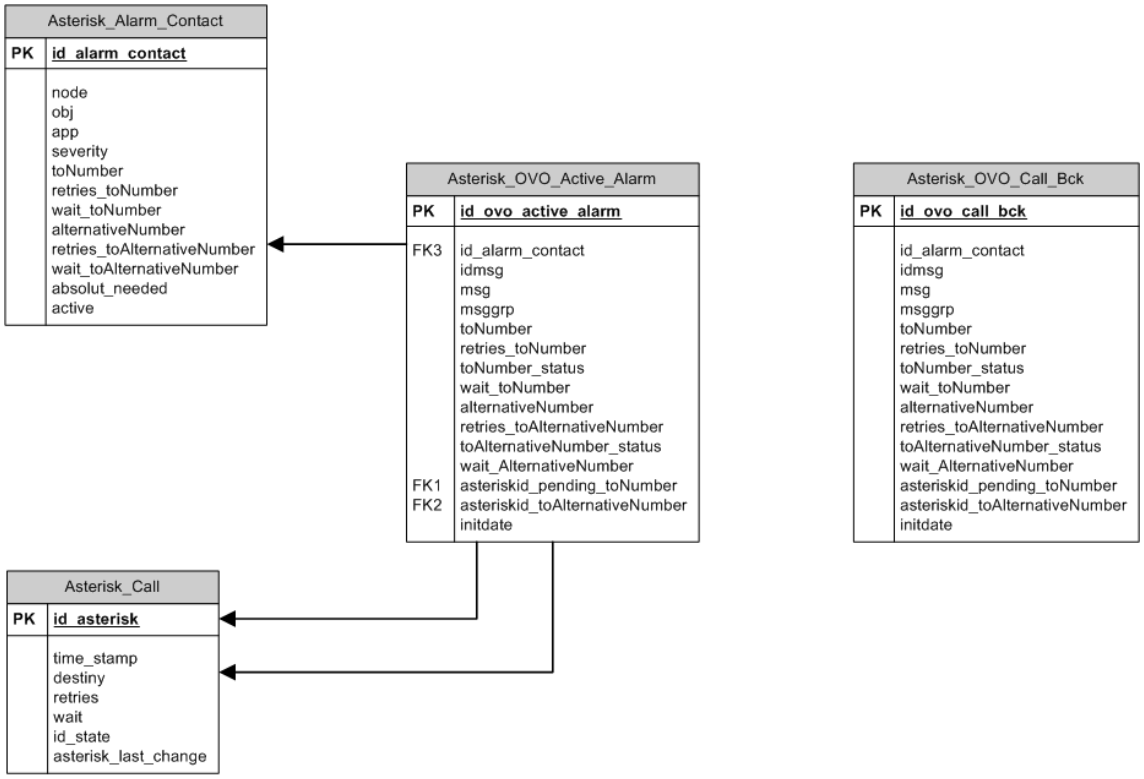


Figura 4.6: Asterisk - Modelo de Dados

A tabela *Asterisk_Alarm_Contact* contém os alarmes mapeados e respectivos contactos, ou seja, os alarmes das equipas com quem já utilizamos o sistema. As quatro primeiras colunas (*node*, *obj*, *app* e *severity*) servem para verificar se os alarmes recebidos têm contactos mapeados. A tabela *Asterisk_OVO_Active_Alarm* recebe os alarmes que chegam ao *HP-OVO* e ficam no estado pendente para serem criadas as chamadas. A tabela *Asterisk_Call* contém as chamadas que estão a ser efectuadas, sendo que o *id_state* vai sendo alterado conforme a chamada é ou não atendida. A tabela *Asterisk_OVO_Call_Bck* será utilizada pelo processo de *housekeeping*. O *id_state* é um enumerável definido na aplicação *C#* que indica qual o estado da chamada (pendente, em progresso, sucesso, falhou, sem contactos, entre outros).

## 4.4 Aplicação Asterisk

Esta aplicação é que contém o fluxo central da plataforma, pois é quem interliga a infor-mação da base de dados onde chegam os alarmes do *HP-OVO* e o *PBX* (*VM* onde está a correr o *AsteriskNOW*).

O fluxo de funcionamento da aplicação começa com a verificação, na tabela *Asterisk_Ovo_Active_Alarm*, da existência de novos alarmes (estado pendente). Em caso afirmativo, verifica se existem contactos para cada alarme (tabela *Asterisk_Alarm_Contact*), se não existir, o estado do alarme é passado para 'sem contactos' e o alarme passa a ser ignorado pela aplicação. Se existe pelo menos um contacto, a aplicação muda o estado para 'em progresso' e cria um registo na tabela *Asterisk_Call* para o contacto de prevenção do alarme, no estado pendente. Depois verifica se existem chamadas para efectuar na tabela *Asterisk_Call*, criando o *cal file*, com o contacto, para o servidor e muda o estado do registo na tabela para 'em progresso'.

O terceiro passo consiste em verificar se já existe resposta para as chamadas em curso, caso a chamada tenha falhado decrementa o número de tentativas e cria um novo registo no estado 'pendente' na tabela *Asterisk_Call*, em caso de sucesso altera o estado para 'sucesso'. As respostas consistem em ficheiros com informação de sucesso ou insucesso que são colocados num *share* entre a aplicação *Windows* e o *script Unix*.

Se uma chamada falhar todas as tentativas para o contacto principal, verifica se existe contacto de escalamento e cria uma chamada para esse numero se aplicável, fazendo o ciclo descrito anteriormente. Na figura 4.7 é mostrado o fluxo macro da aplicação.

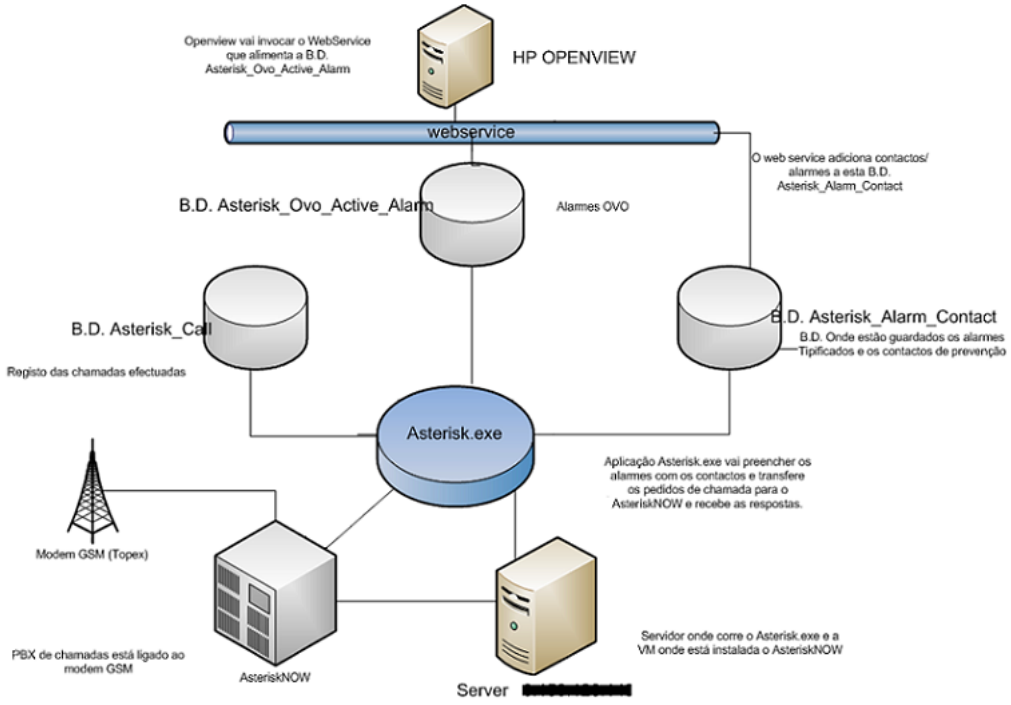


Figura 4.7: Asterisk - Fluxo do sistema

## 4.5 Aplicação Web (CallMonitor)

A aplicação *Web* foi desenvolvida dentro do projecto piloto e portanto disponibiliza apenas informação, para consulta, referente às chamadas (figura 4.8). Como se pode ver na imagem, as chamadas foram todas efectuadas com sucesso para o contacto da prevenção, por esta razão o estado da chamada para o contacto de escalamento fica no estado inicial (pendente).

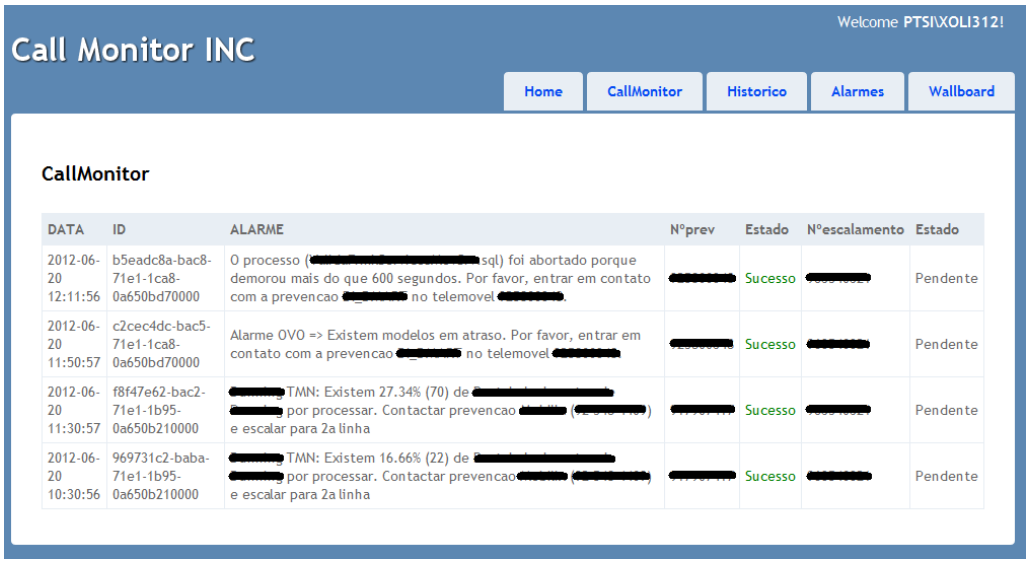


Figura 4.8: CallMonitor - Chamadas em espera

Neste aplicação, também é possível, consultar o histórico das chamadas (figura 4.9), neste menu podemos ver todas as chamadas que já foram efectuadas, incluindo os estados das chamadas para as prevenções e escalamentos. As chamadas ficam visiveis na vista do *CallMonitor* até ao final de cada turno da Produção OA (8hrs), no final de cada turno existe um processo que as remove da vista que é apresentada nas secções *CallMonitor* e *Wallboard* e passam a ser disponibilizadas nesta secção.

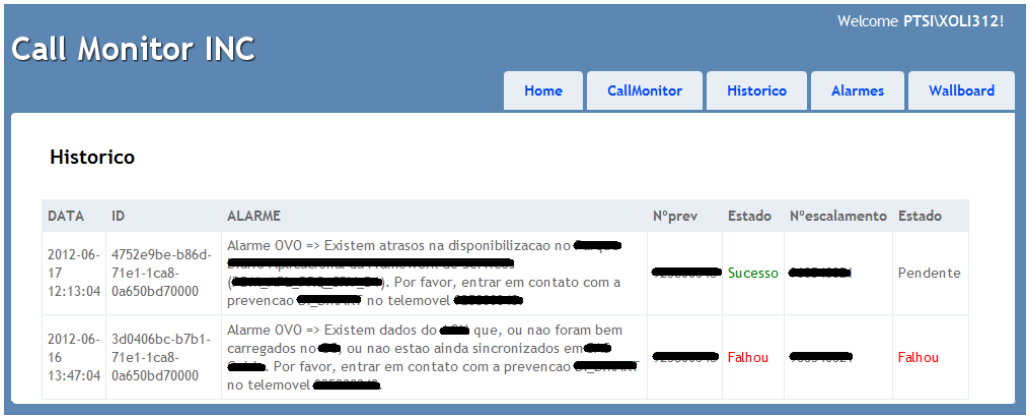


Figura 4.9: CallMonitor - Histórico

Pode-se consultar, ainda, os alarmes tipificados e respectivos contactos de prevenção e escalamento (não é mostrada imagem devido à quantidade de informação confidencial) e ainda uma secção, *Wallboard* (figura 4.10), com a informação sobre as chamadas realizadas durante o turno, colocando cores verde, vermelho ou cinzento, em caso de sucesso, insucesso ou chamada em espera, respectivamente. Esta vista foi criada para ser disponibilizada num monitor (*wallboard*) e assim ser vista mais facilmente pelos elementos de turno.

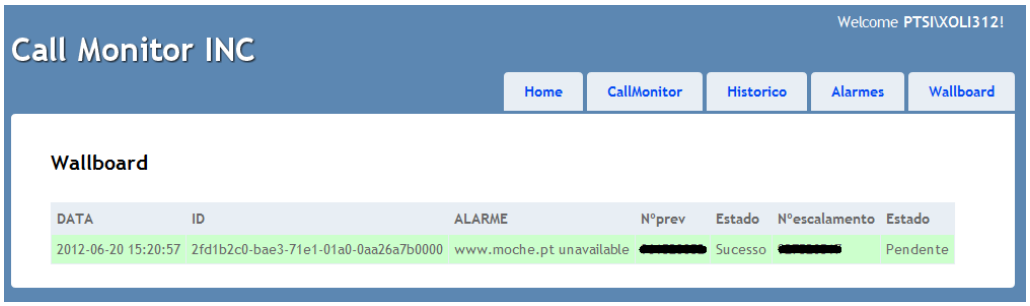


Figura 4.10: CallMonitor - Wallboard

## 4.6 Integração com outras Aplicações (Webservice)

O *AsteriskWS* foi desenvolvido com o objectivo de alimentar, com alarmes despoletados por várias ferramentas, a BD do *Asterisk*. No entanto, a ferramenta oficial da PT, conforme já foi dito atrás, é o *HP-OVO* e portanto, a integração central do *Asterisk* é com esta ferramenta.

Para poder alimentar a circuito do *Asterisk*, trabalhei em conjunto com a equipa de suporte ao *HP-OVO*, de forma a criar um método que pudessem chamar e adicionar os alarmes à minha BD. Assim aconteceu e, sempre que chega um alarme ao *HP-OVO* para o grupo da Produção OA, é invocado o *AsteriskWS*, mesmo que não seja necessário efectuar uma chamada, daí a necessidade de ter uma tabela com alarmes tipificados, para saber quais devo criar uma chamada automática.

O *Asterisk* encontra-se integrado também, com a ferramenta de monitorização interna da OA, a *Mon-IT*. Como esta ferramenta tem outros meios de envio de alarmes, para além do *HP-OVO*, e alguns desses alarmes podem ser críticos, foi decidido integrar as duas aplicações.

O *AsteriskWS* possibilita a integração com qualquer aplicação que tenha a necessidade da realização de uma chamada automática. Pode ter, inclusive, outro tipo de utilização para além do *report* de alarmes.

Para além das integrações, o *AsteriskWS* também possibilita o registo de alarmes tipificados na tabela *Asterisk_Alarm_Contact*, como será discutido no trabalho futuro, esta funcionalidade passará para a aplicação *Web*.

# Capítulo 5

# Projecto Checklist

## 5.1 Contextualização

A *Checklist* é uma aplicação, interna, de suporte à OA, que consiste numa base de dados de informação relevante para as equipas da OA. Já existe uma *CMDB* federada da PT, no entanto como não é gerida pela OA e é uma aplicação fechada, existem sérias dificuldades em adicionar informação à mesma. Esta situação foi fulcral para a criação da *Checklist*, onde podemos facilmente aceder e alterar informação relevante para a OA, vinda da CMDB.

O modelo de dados foi desenhado de maneira a ser o mais escalável e flexível possível, tendo sido criadas as noções de *CI* (*Configuration Item*) genérico e relações genéricas. Um *CI* genérico é uma entidade que pode representar desde uma pessoa a uma memória RAM dentro de um servidor, pode ser aplicacional ou estrutural. É composto por um tipo e por um conjunto de propriedades associadas a esse tipo, por exemplo, nome, morada, memória, processador. A noção de Relação Genérica, pretende generalizar qualquer relação que possa existir e ser necessária entre *CIs*, como por exemplo, 'pertence a', 'responsável por' ou 'dependente de', ou mesmo outro tipo de relação. Com esta lógica podemos representar o que quisermos. Para garantir relações correctas também foi criada a noção de Relação Permitida entre *CIs*.

Estas configurações são todas possíveis a partir de uma aplicação Web com interface simples e dinâmico, de forma a disponibilizar toda a informação necessária de forma rápida e legível.

Sendo um repositório de informação, tornou-se na plataforma central da OA pois consegue disponibilizar qualquer tipo de informação às outras plataformas construídas por nós. A principal informação que é disponibilizada serve para completar a informação de alarmes pois é possível obter toda a informação sobre quais os impactos aplicacionais, estruturais e de negócio que essa situação provoca.

Esta plataforma envolve tecnologias, como bases de dados *SQL Server*, aplicações *Web* em *ASP.NET MVC 3*, *webservices* em *C#* recebendo e respondendo a pedidos em

*XML*, *LINQ* para executar comandos *SQL* na BD, entre outras.

## 5.2 Primeira Abordagem vs Modelo Actual

Inicialmente a Checklist foi focada principalmente para informação relacionada com pessoas (*contacts*), equipas (*teams*) e sistemas (*systems*), sendo este último genério o suficiente para conter vários tipos de sistemas (aplicação, servidor, posto de trabalho, etc), continha também informação sobre problemas e suas resoluções/escalamentos. As relações foram separadas em par, responsável e atribuído. A relação Par (*Pair*) relacionava as equipas com os seus elementos, a relação Responsável (*Owner*) que relacionava as equipas ou pessoas responsáveis por sistemas e a relação Atribuído (*Assignee*) que relacionava problemas com as equipas ou pessoas a quem a operação os devia reportar.

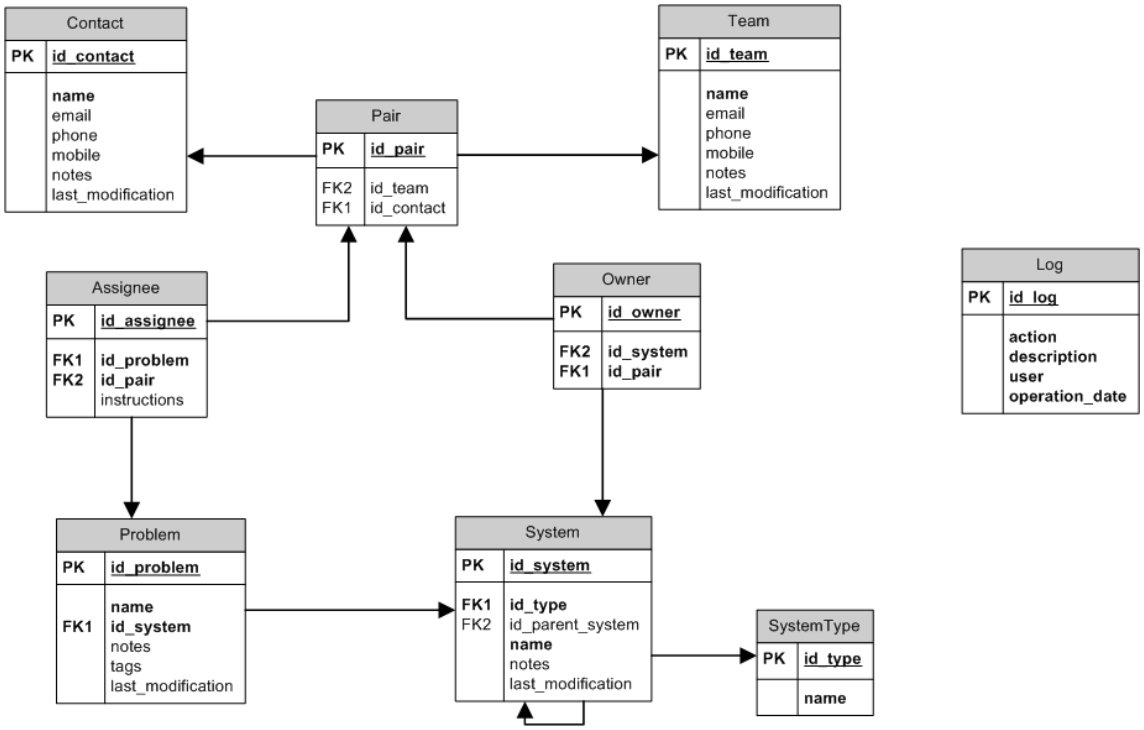


Figura 5.1: Checklist - Primeiro Modelo de Dados

Rapidamente apercebi-me que este modelo de dados era pouco escalável para as necessidades que vieram sendo definidas no decorrer do trabalho na empresa e também algo confuso devido à forma como as várias relações foram definidas, inicialmente, como se pode ver na figura 5.1.

Após uma nova análise desenhei modelo de dados actual 5.2 que foi pensado de forma a ser o mais escalável possível e as relações não serem tão confusas.

Este novo modelo baseia-se em *CIs* (*Chklst_CI*) que podem ser de vários tipos (*Chklst_CI_Type*). Cada tipo de *CI* contém uma ou mais propriedades (*Chklst_Property*). A

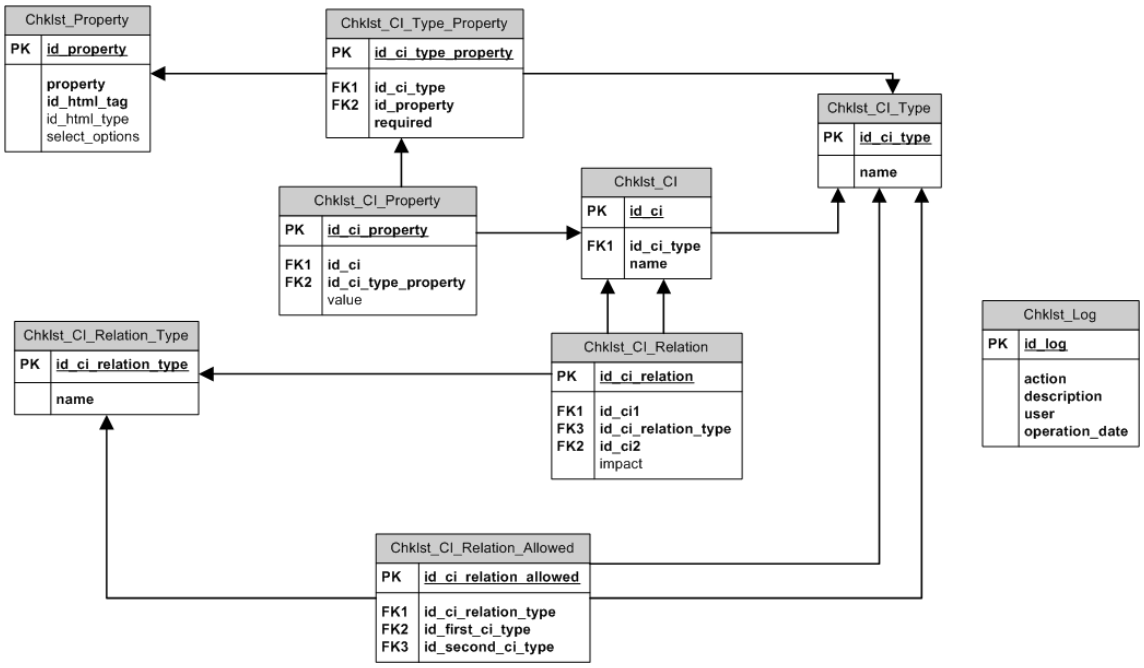


Figura 5.2: Checklist - Modelo de Dados actual

forma que pensei para relacionar as propriedades e os *CIs* foi a seguinte. A tabela *Chklst_CI_Type_Property* contém a relação entre tipo de *CI* e as suas propriedades e se estas são obrigatórias aquando da criação do *CI*. A tabela *Chklst_CI_Property* serve para guardar o valor da propriedade de um *CI* específico.

Até aqui mostrei como um *CI* é composto, agora vou mostrar como se relacionam os *CIs* entre si. Existem vários tipos de relações (*Chklst_Relation_Type*) que os *CIs* podem utilizar, sendo que as relações entre *CIs* específicos ficam guardadas na tabela *Chklst_CI_Relation*, que também guarda o impacto (valor numérico) que essa relação tem no negócio, em caso de falha. Como os *CIs* são objectos genéricos, é necessário garantir que as relações entre si estejam em conformidade com a realidade (*Chklst_CI_Relation_Allowed*), por exemplo, Equipa (CI) 'responsável por' (relação) Processo Aplicacional (*CI*), esta é uma relação válida, mas já não faz sentido existir a relação contrária.

A tabela *Chklst_Log* foi desenhada para manter um arquivo das acções efectuadas via Web e que as executou, nomeadamente adições de novos registos ou alterações aos dados dos registos existentes.

## 5.3 Gestão de CIs (Aplicação Web)

Todo o processo de criação e alteração de *CIs* é efectuado através de uma aplicação *Web* que contém duas componentes, uma de gestão (figura 5.3) e uma de disponibilização da informação através de pesquisa (figura 5.4).

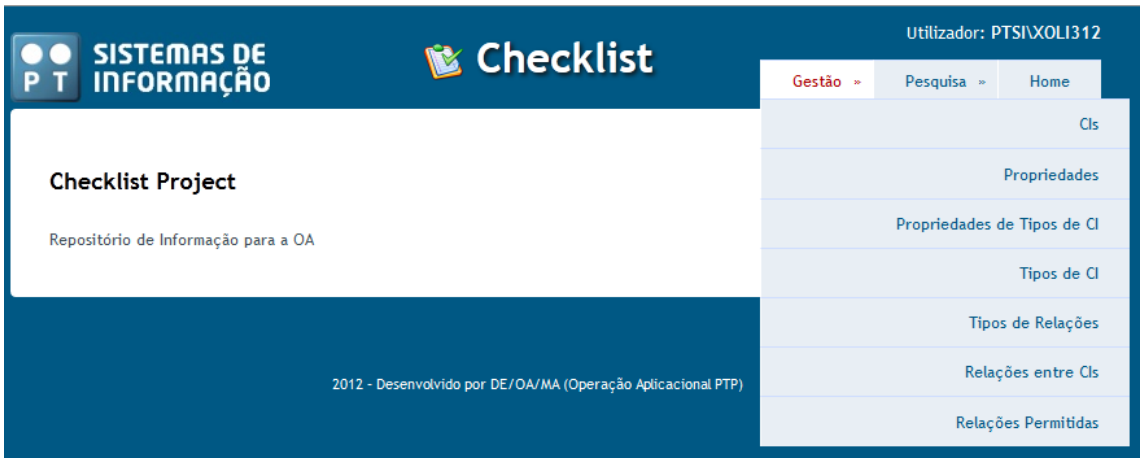


Figura 5.3: Checklist - Gestão

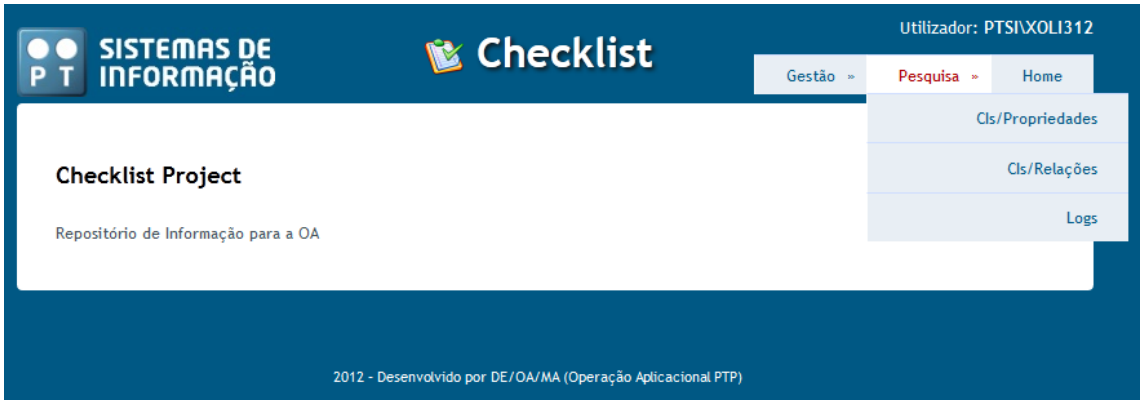


Figura 5.4: Checklist - Pesquisa

A componente de gestão permite de forma rápida e fácil criar, editar e listar *CIs*, tipos de *CIs*, relações entre *CIs* ou propriedades de *CIs* e ainda relacionar todos estes conceitos, ou seja, adicionar/remover propriedades a um tipo de *CI*, relacionar dois CIs, etc. Esta componente é só para ser usada por elementos de 2ª linha da OA.

Conforme foi dito na secção acima, os *CI* têm um Tipo, e cada Tipo tem associado uma ou várias propriedades.

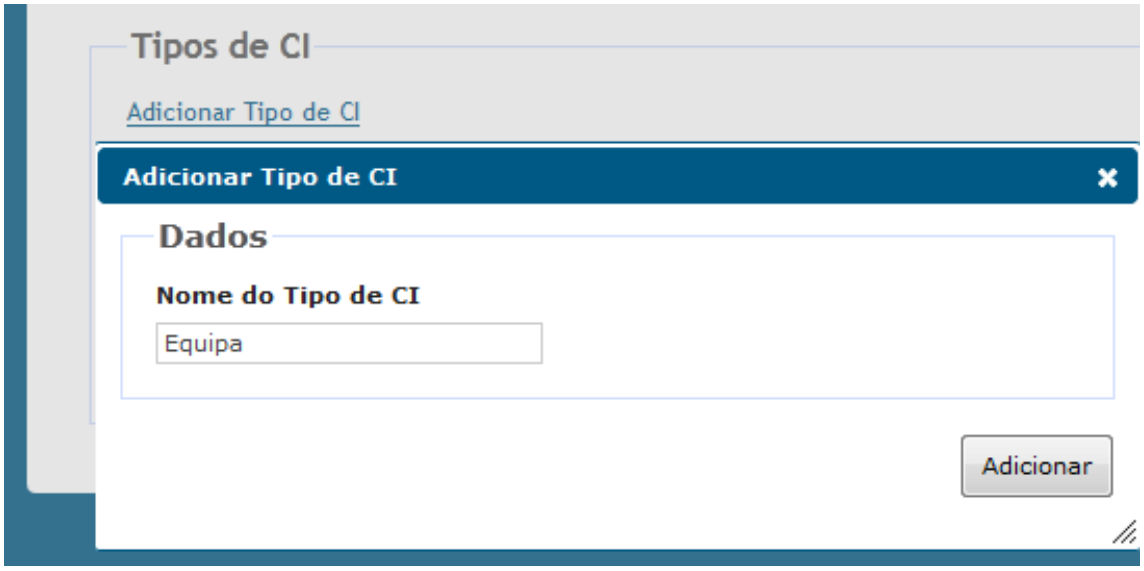


Figura 5.5: Checklist - Adicionar Tipo de CI

Na figura 5.5 é mostrada a página (*popup*) de criação de uma Tipo de *CI* e por trás a listagem dos Tipos de *CI*, onde se pode ver um *link* para a página que contém a lista das suas propriedades e a possibilidade de adicionar mais.

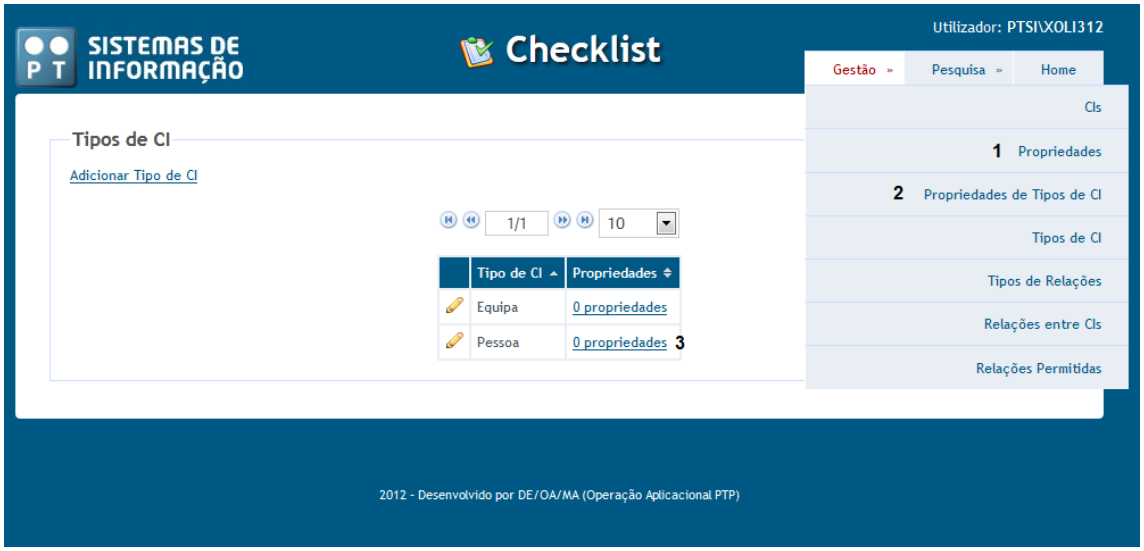


Figura 5.6: Checklist - Adicionar Propriedade a Tipo de CI

Na figura 5.6 estão assinaladas as formas de adicinar uma ou mais propriedades a um Tipo de *CI*. Primeiro é necessário criar a propriedade (1) (figura 5.7). Após criar a propriedade é necessário adiciona-la ao Tipo de *CI* que se pretende (2) (figura 5.8).

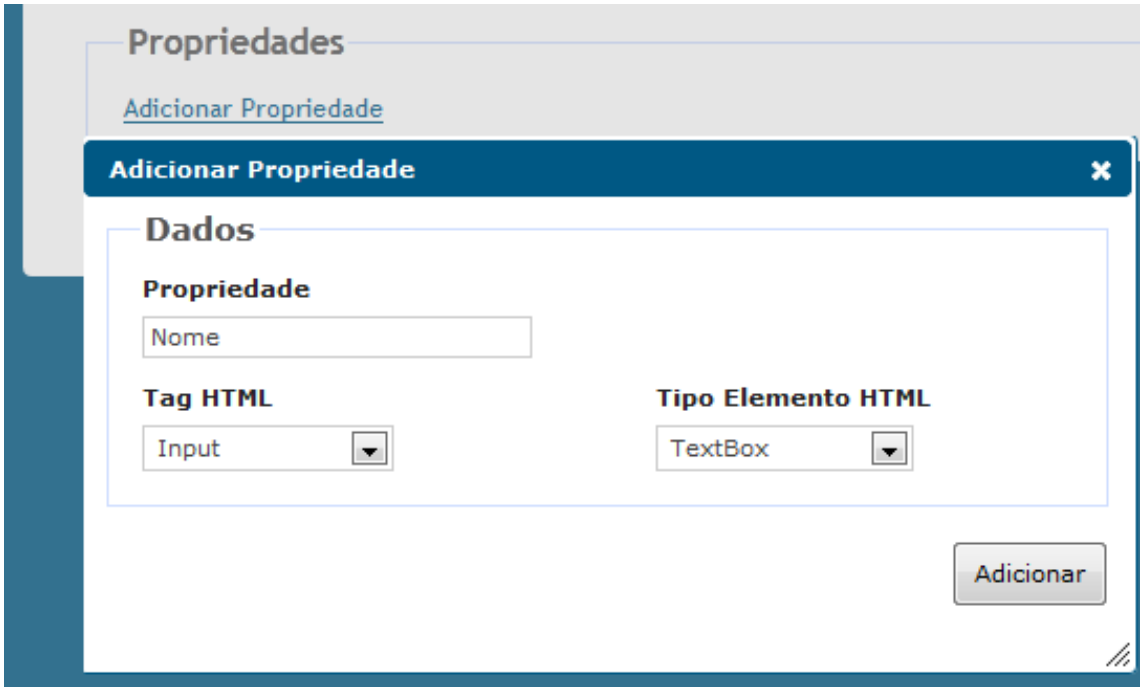


Figura 5.7: Checklist - Adicionar Propriedade (1)

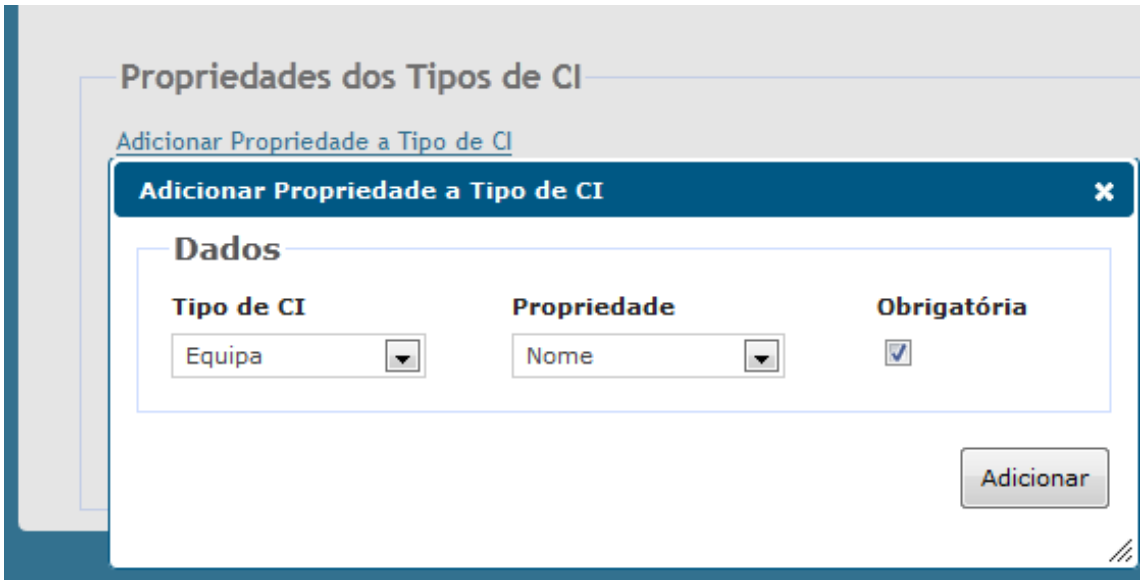


Figura 5.8: Checklist - Adicionar Propriedade a Tipo de CI (2)

O *link* assinalado por (3) (figura 5.9) substitui os outros dois passos. Nesta página existe uma lista com propriedades pré-configuradas mas sem associação ao Tipo escolhido. Para adicionar basta escolher a propriedade e carregar em Adicionar.

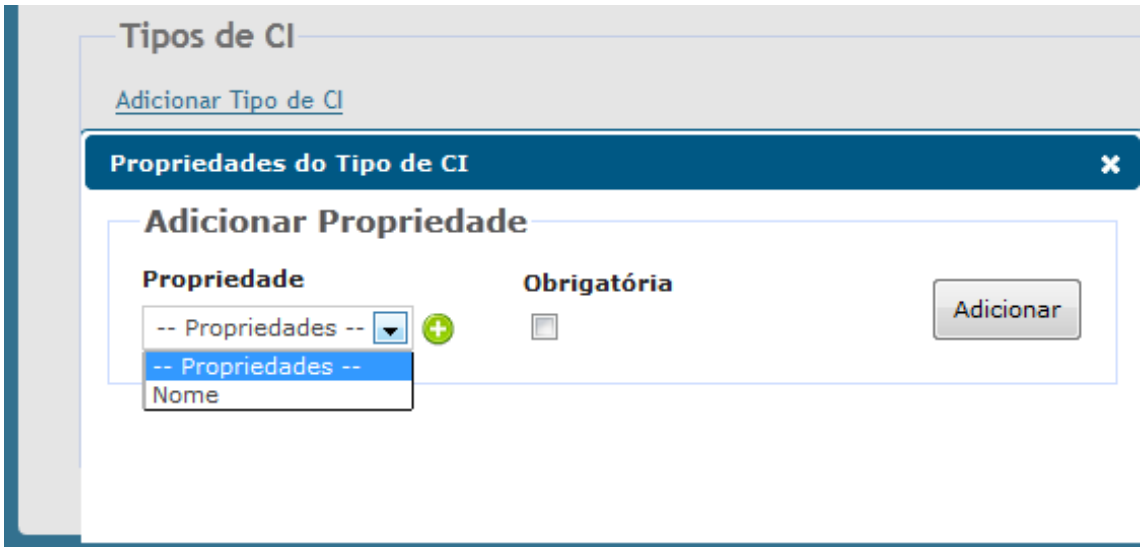


Figura 5.9: Checklist - Adicionar Propriedade a Tipo de CI (3)

Caso o utilizador queira adicionar uma nova propriedade, ou seja, que ainda não exista na BD, então tem o botão verde com o símbolo de adição (+) onde aparece um novo *popup* (figura 5.10) para o utilizador poders criar uma nova propriedade e adicioná-la automaticamente ao Tipo de *CI* escolhido.

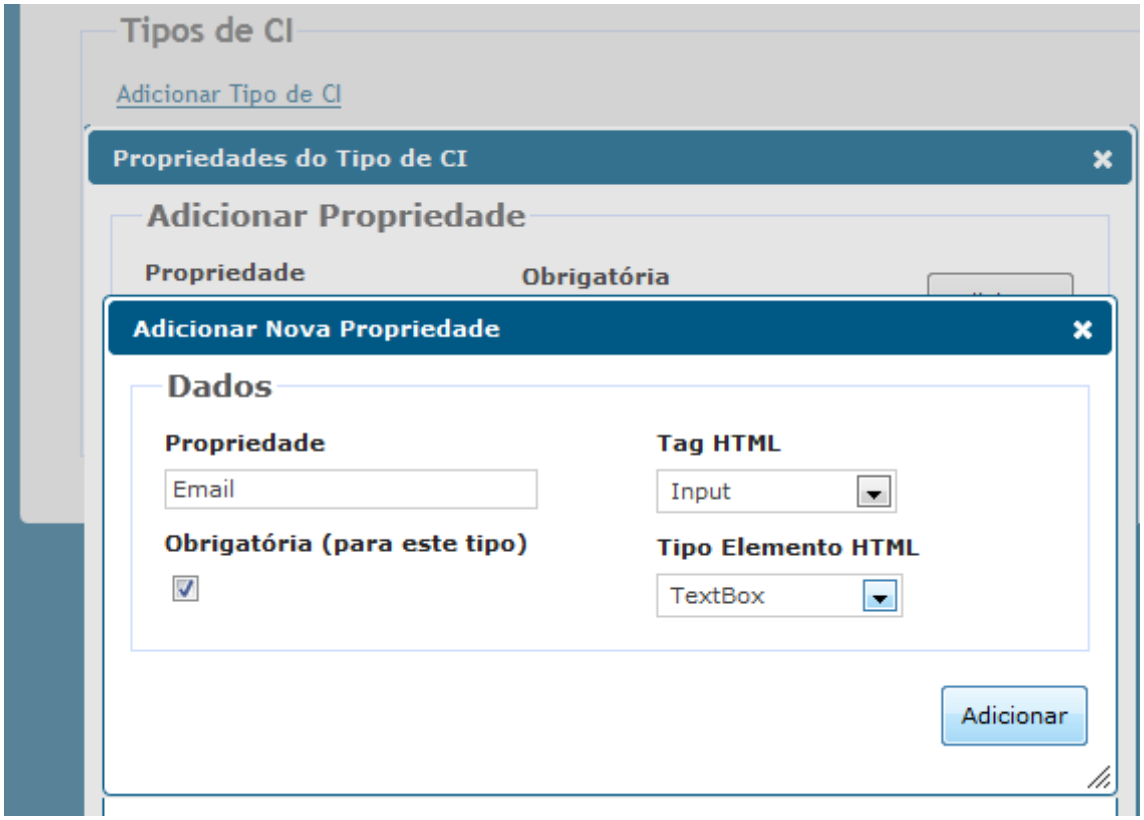


Figura 5.10: Checklist - Adicionar nova Propriedade a Tipo de CI

Na criação de *CIs* todo o processo tem de ser dinâmico após ser escolhido o tipo do *CI* (figura 5.11), pois cada um tem as suas propriedades. Para isso recorri ao *javascript* para adicionar dinamicamente à página os elementos *html*, referentes às propriedades. Cada propriedade tem guardada a *tag html* correspondente e caso seja a *tag input* também é guardado o seu tipo (*textbox*, *checkbox*, etc).

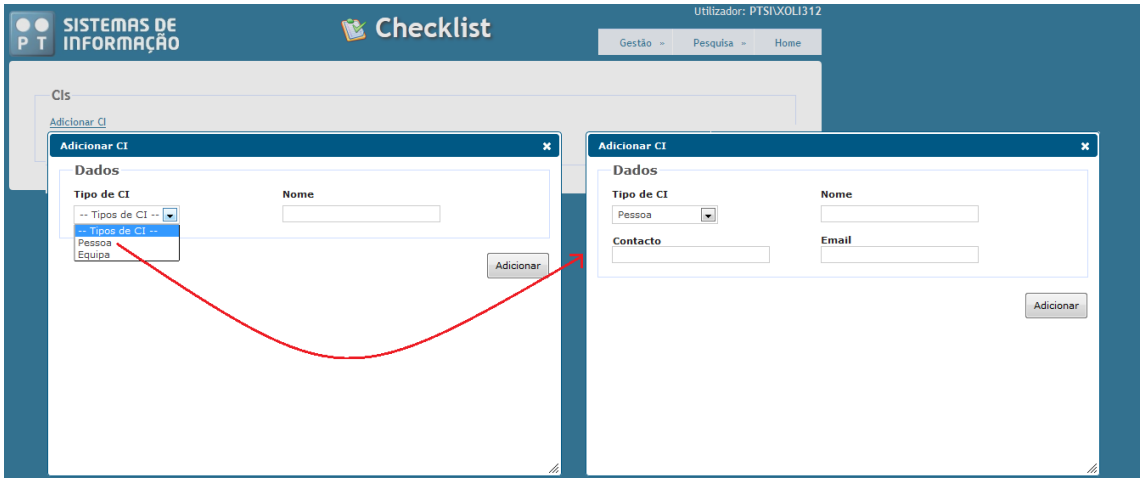


Figura 5.11: Checklist - Adicionar CI

Na página principal é mostrada a lista com os *CIs* criados e um *link* para gerir as suas relações com os outros *CIs* (figura 5.12).

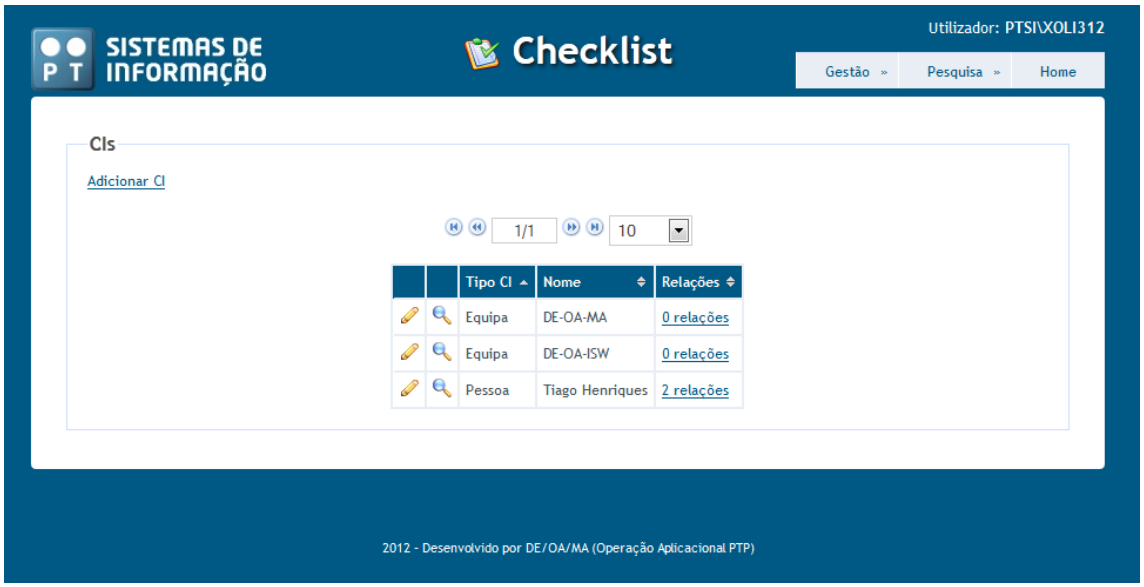


Figura 5.12: Checklist - Lista de CIs

Para além da criação do *CI* também é necessário disponibilizar os restantes *CIs* relacionáveis. Para tal coloquei um *link* na lista dos *CIs* para um *popup* onde é mostrada uma lista com as relações já existentes para o *CI* que se está a consultar e também a possibilidade de adicionar novas relações com outros *CIs* (figura 5.13).

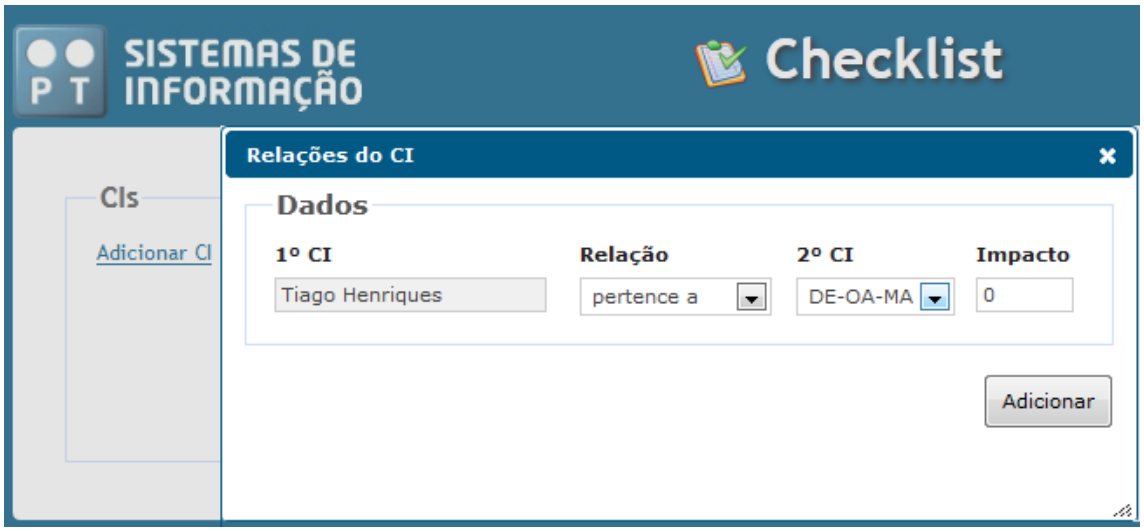


Figura 5.13: Checklist - Adicionar Relação entre *CIs* (lista de CIs)

Outra maneira de adicionar relações entre *CIs* é aceder via menu à secção 'Relações entre CIs' (figura 5.14).

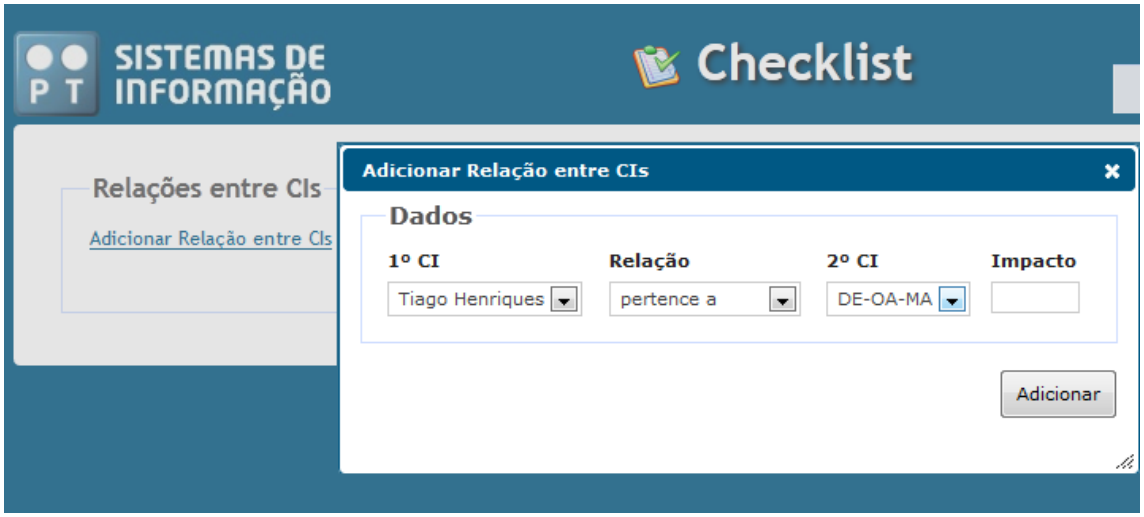


Figura 5.14: Checklist - Adicionar Relação entre CIs (menu)

Para criar os tipos de relações basta aceder à secção 'Tipos de Relações' (figura 5.15).


Figura 5.15: Checklist - Adicionar Tipo de Relação

De forma a garantir que não são inseridas relações ilógicas, são criadas Relações Permitidas entre Tipos de *CI* (figura 5.16). Desta forma quando um utilizador está a relacionar dois CIs, só é possível criar relações que estejam na tabela das Relações Permitidas.


Figura 5.16: Checklist - Adicionar Relação Permitida

Na secção de pesquisa avançada, extremamente rápida e eficaz, é possível aos elementos das equipas da OA, 1ª ou 2ª linha, pesquisar informação relevante ao seu trabalho. Foi efectuada uma separação para simplificar a disponibilização da informação. Numa página

é possível pesquisar por *CIs* e as suas Propriedades (figura 5.17), na outra é possível pesquisar por *CIs* e as suas relações (figura 5.18). Esta segunda página de pesquisa é a mais importante dado que as relações entre os *CIs* podem ter impactos elevados, nomeadamente a nível de negócio, para tal quando existe um problema num *CI* é necessário saber quais os *CIs* afectados e os respectivos impactos. Desta forma a informação consegue ser facilmente disponibilizada, incluindo a percentagem do impacto que um problema, num *CI*, pode implicar. (Este cálculo ainda não é apresentado pois ainda está a ser analisado um modelo matemático que o permita calcular de forma mais próxima da realidade).

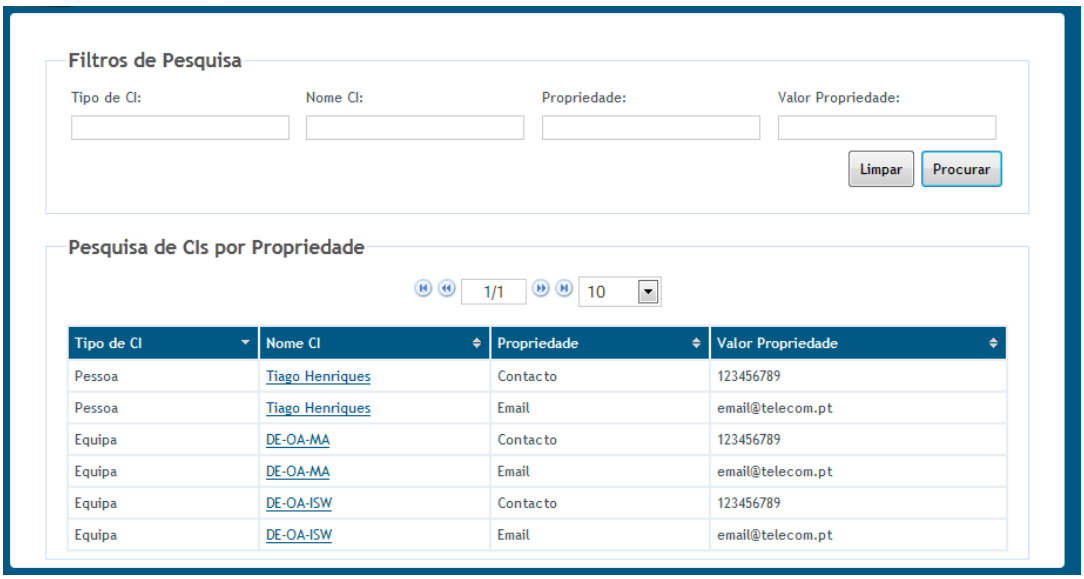


Figura 5.17: Checklist - Pesquisa CIs/Propriedades

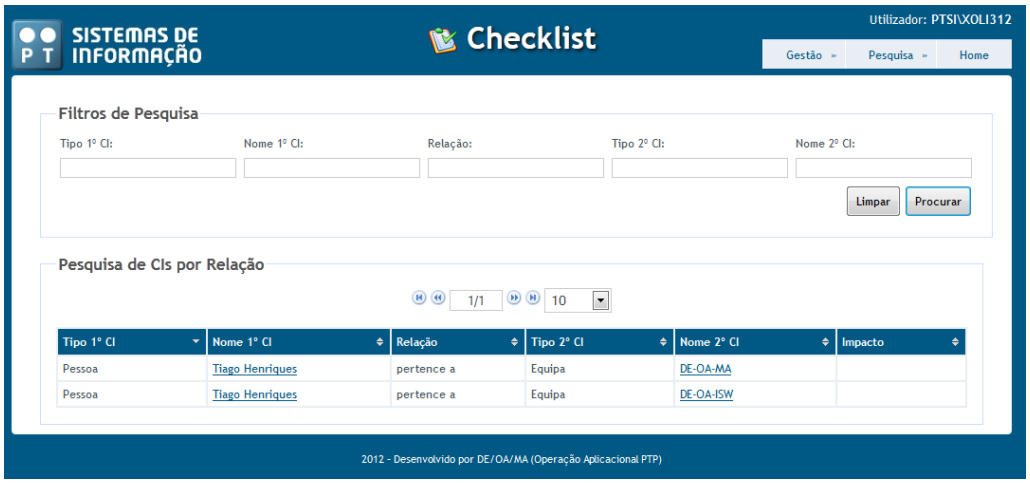


Figura 5.18: Checklist - Pesquisa CIs/Relações

Também é possível consultar os logs das acções efectuadas via *Web* e quem as fez (figura 5.19).

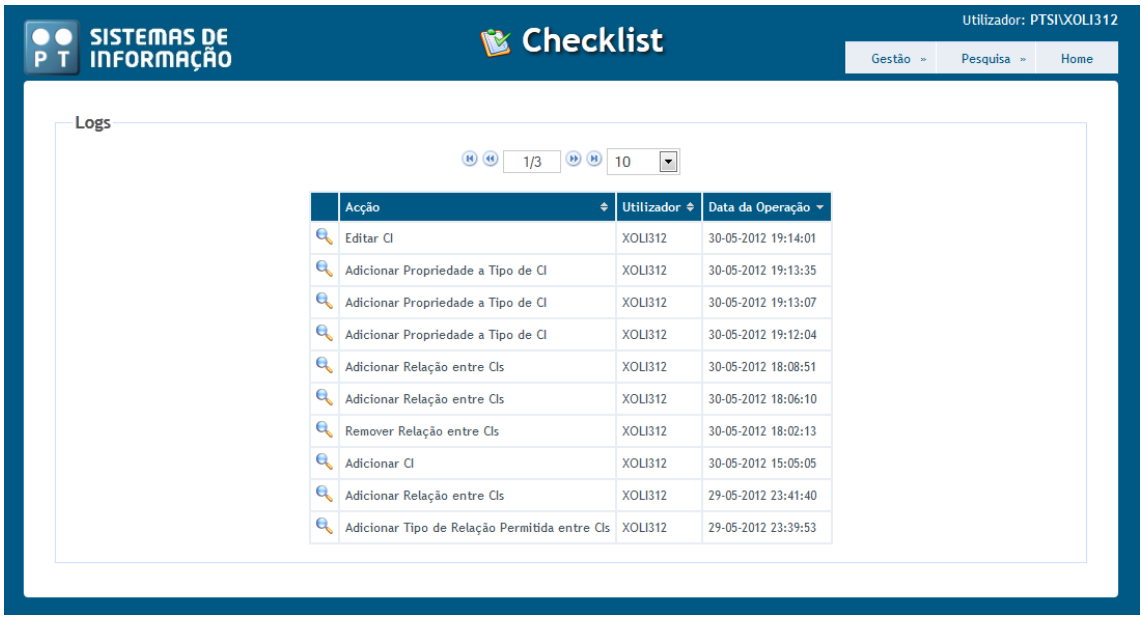


Figura 5.19: Checklist - Pesquisa Logs

## 5.4 Carregamento Massivo de Dados

Devido a OA ser uma área transversal à PT, necessita de bastante informação sobre equipas, elementos, sistemas associados, etc.

Para facilitar a aquisição dessa informação, criei um ficheiro *Excel* para servir de template, com várias *sheets*, desde o Tipo de *CI*, às relações entre CIs. Este tem sido distribuído pelas várias equipas para preencherem os seus contactos e sistemas que são responsáveis.

Para criar o ficheiro *Excel* utilizei a biblioteca *Microsoft.Office.Interop.Excel*. Para além das várias *sheets* com cabeçalhos para o utilizador saber o que está a preencher, também criei macros em *VBS*, que são inseridas no ficheiro, também no *C#*, para quando um utilizador está a preencher, por exemplo, os campos de um *CI*, poder escolher outros *CIs* para criar relações entre os mesmos.

## 5.5 Integração com outras Aplicações (Web Service)

A Checklist pretende ser a base central de informação de toda a área da OA. Para possibilitar esta centralização é necessário integrar a aplicação com todas as ferramentas desenvolvidas pela OA para que seja possível utilizar essa informação.

Conforme já foi dito, a *Checklist* não pretende substituir a *CMDB* mas sim ser uma camada complementar, paralela e independente. Para isso as equipas que gerem a CMDB forneceram-nos acesso ao *Webservice* próprio para podermos aceder à sua informação.

Esta integração serve para quando um utilizador pretende informação extra sobre um *CI* que não é apresentada na *Checklist*, como por exemplo custos aplicacionais ou relações com outros CIs, que à partida são irrelevantes para a OA. A informação adicional vinda da *CMDB* é apenas apresentada e não inserida na BD da *Checklist* mas ao inserir um *CI* é possível pedir à *CMDB* as suas relações existentes

A integração com as aplicações internas da OA é feita através de *Webservices* que recebem pedidos em formato *XML* e devolvem o resultado no mesmo formato. Desta forma uma aplicação que necessite, por exemplo, do contacto de uma equipa, faz um pedido ao *WS* com o nome da equipa e recebe a informação pretendida. Isto é necessário, por exemplo, no caso da *Mon-IT* ou do *Sputnik* para incluírem a informação sobre os responsáveis pelo *CI* e também o impacto que um certo *CI* tem no negócio.

Para integração com a *Mon-IT* e *Sputnik* foi criado um método que recebe um pedido *XML* com o nome de uma máquina e/ou monitorização e retorna os contactos das pessoas e/ou equipas responsáveis pela máquina/monitorização.

A *Checklist* também se encontra integrada com o Medusa (aplicação de gestão de indisponibilidades), em que o pedido contém o nome de uma máquina e/ou aplicação e o *Webservice* devolve os impactos que a máquina/aplicação tem a nível de negócio. Isto é importante para saber quais as áreas que irão ser afectadas com a intervenção, de maneira a encontrar a melhor altura para a realizar, diminuindo assim o seu impacto.

Sendo a *Checklist* uma aplicação central, não precisa de se preocupar com a informação que tem de fornecer, a quem cabe essa responsabilidade é a aplicação que pretende a informação. A partir do momento em que uma aplicação válida faz uma chamada válida ao *Webservice* é sempre devolvido um resultado.

Os pedidos ao *Webservice* têm de estar identificados com o nome da aplicação requerente, de seguida *tags* com o tipos de *CIs* e os nomes do *CIs* pretendidos. A resposta do *Webservice* é devolvida no mesmo formato, sendo que as tags estão identificadas com o nome do *CI* e contém os valores das propriedades ou relações dos CIs.

## 5.6 Documentação

A documentação da *Checklist* é composta por um manual de criação/gestão de *CIs* e outro para comunicação com o *Webservice*, onde é explicado o formato do pedido a ser enviado e da resposta recebida. Criei ainda um manual de utilização da aplicação que compila e lê o template de carregamento massivo de dados e como se deve preencher o mesmo.

# Capítulo 6

# Gestão

## 6.1 Contextualização

Dentro da área OA existem equipas de 1ª e 2ª linha. Às equipas de 1ª linha chegam todo o tipo de pedidos de várias áreas, sendo que uns são logo tratados e outros encaminhados para as equipas de Suporte Técnico. As equipas de 2ª linha têm uma forte componente de *coaching*, *backup* técnico de identificação e resolução de incidentes complexos nos sistemas aplicacionais, formação de novos elementos, entre outros.

É neste contexto que entra a componente de gestão no meu PEI, pois para além do trabalho inerente à equipa de Monitorização Aplicacional onde apliquei e desenvolvi os meus conhecimentos técnicos, também consegui aplicar e desenvolver os meus conhecimentos a nível de gestão, devido à minha maturidade e experiência dentro da área.

## 6.2 Formação

Os meus conhecimentos, a nível de *C#*, *.Net*, *SQL Server* e *Web*, no inicio do meu PEI, eram apenas académicos. Numa primeira instância comecei a estudar as linguagens que me sentia menos confortável e só depois comecei a desenvolver os projectos.

Durante este tempo também recebi formação sobre aplicações de monitorização corporativas, entre as quais o *BSM* (*Business Service Management*) e *SiteScope*, ambas da *HP* (*Hewlett-Packard*).

O *HP BSM* é um conjunto de ferramentas de gestão e monitorização *end-to-end* de serviços em *Data Centers* e infra-estrutura subjacente. No contexto da OA utilizamos esta ferramenta para monitorizar a disponibilidade de aplicações *Web-based* ou outras.

O *HP SiteScope* é uma ferramenta de monitorização focada na monitorização de performance e disponibilidade de infra-estruturas *IT* distribuídas, incluindo servidores, sistemas operativos, redes e serviços de *Internet*, entre outros. Esta ferramenta testa páginas *Web* utilizando monitorização simétrica mas não está limitada a aplicações *Web*. A equipa responsável pela ferramenta disponibiliza várias monitorizações *default* como disponibi-

lidade de bases de dados, espaço livre em *filesystems*, etc. No fim da configuração da monitorização, adicionamos a invocação da política do *HP-OVO* para serem enviados os alarmes.

Para além das formações que recebi, também dei formações a elementos da minha equipa e não só, em *C#*, *shellscript* e *SQL*. Nestas formações foram desenvolvidos alguns *scripts* para ajudar a monitorizar processos de *Unix*, um processo automático (secção6.4). A nível de *SQL*, expliquei noções básicas sobre criação de tabelas e relações entre as mesmas.

## 6.3 Coaching

A minha passagem de 3 anos pela equipa de operação Supervisão TMN (24x7), permitiu-me conhecer transversalmente o negócio da TMN. Com a minha passagem para a equipa de Monitorização Aplicacional também acresci funções de 2ª linha.

Durante o meu PEI entraram vários elementos para a equipa de operação, aos quais passei o meu conhecimento a nível de processos e circuitos monitorizados, procedimentos a efectuar nos pedidos que nos chegam das outras áreas, como questões de portabilidade ou referências MB que não funcionam. Este acompanhamento da 1ª linha não incidiu apenas nas formações de novos elementos mas também num acompanhamento e *backup* técnico, diários.

No seio da minha equipa também entraram novos elementos e aqui também existiu uma formação e acompanhamento cuidados, explicando os procedimentos e o funcionamento das ferramentas que utilizamos na vertente de Instalação de *Software* e mostrei as plataformas que utilizamos para desenvolver *software* na vertente de Monitorização Aplicacional.

## 6.4 Automatizações

Para além do acompanhamento a nível de resolução de incidentes, também automatizamos processos que pedem à 1ª linha para fazer manualmente, implementação de alarmes, *reporting*, etc. Nesta vertente de *coaching* desenvolvi um processo automático chamado *Reports Flash*. Este foi apenas o primeiro de muitos que pretendo automatizar e serviu como caso base nas formações de *C#*.

Todos os meses a dia 15 a Produção OA acedia a um *website*, fazia o *download* de 6 ficheiros *Excel* para o disco local. Após ter os ficheiros, era necessário preencher um template, também em *Excel*, copiando cada ficheiro para a *sheet* correspondentes. No final comprimiam o template e disponibilizavam o ficheiro *.zip* no *sharepoint* corporativo.

Após falar com a equipa que disponibilizava os ficheiros no *website*, começaram a enviar-nos por *email* para uma *mailbox* criada para processos automáticos. Quando re-

cebemos os *emails*, existe uma *macro* que copia os ficheiros para um *share* e criei um programa em *C#* para ler os ficheiros do *share*, criar o *template* com as *sheets* preenchidas, comprimir e enviar para o *sharepoint*. O programa *C#* utiliza, a nível de tecnologia, bibliotecas com a *Microsoft.Office.Interop.Excel* para criação, leitura e escrita em ficheiros *Excel*, *SharpZip* para a compressão de ficheiros e ainda a ligação com o *Webservice* do *Sharepoint* corporativo da PT.

O processo corre todos os dias (1 vez) através de uma *schedule task* e caso os ficheiros não se encontrem no *share*, pois podem chegar em dias diferentes, o programa termina sem processar nada.

# Capítulo 7

# Conclusão

## 7.1 Discussão

Todos os projectos que desenvolvi foram sempre a pensar na eficiência e no utilizador final, fazendo os interfaces simples e práticos e o acesso à informação o mais rápido possível e sem entropias. Os requisitos estiveram em constante mudança, portanto todas as aplicações são altamente escaláveis e abertas para qualquer interligação com outras aplicações que for necessária.

Conforme pode ser visto nas discussões abaixo, com a continuação destes projectos irei efectuar algumas alterações e integrações já previstas. No entanto, as integrações continuarão para além do PEI devido ao constante aumento da visibilidade da minha equipa.

Continuarei a automatizar processos e a melhorar cada vez mais a camada de monitorização aplicacional e infra-estrutural da PT.

### 7.1.1 Sputnik

Quando comecei o PEI, o *Sputnik* já se encontrava aplicado em *5* circuitos de monitorização e já tinha uma certa visibilidade dentro da PT. Com a entrada desta nova versão já foram criados mais 9 circuitos de diferentes áreas, incluindo alguns domínios aplicacionais de parceiros internacionais, como a TT (Timor Telecom) e CVT (Cabo Verde Telecom). A rapidez e a facilidade como se monta uma nova instância no novo *Sputnik* veio consolidar a sua posição dentro da OA, mas principalmente aumentou, exponencialmente, a sua visão e a da OA dentro da PT.

A integração com a *Mon-IT* possibilitou a reutilização de monitorizações, podendo estas serem partilhadas pelas duas aplicações, retirando assim a necessidade da existência de outros processos a fazerem a mesma monitorização. Possibilitou assim que, em simultâneo seja enviado um alarme via *Mon-IT* e actualizado o estado da respectiva sonda no *Sputnik*. Desta forma a identificação do problema e impactos associados, é feita de

forma imediata e consequentemente leva a uma visão rápida da solução a aplicar, permitindo assim um cumprimento de *SLAs* e uma minimização brutal do impacto no cliente.

Com o aumento exponencial do reconhecimento que a minha equipa está a ter dentro da OA e, especialmente fora dela, é esperado que o número de instâncias aumente e também a possibilidade de aplicar o *Sputnik* a outros domínios e possivelmente a clientes da PTSI.

#### 7.1.1.1 Resultados

Desde que a aplicação entrou em produção consegui retirar alguns resultados extraídos da base de dados e compará-los com a versão anterior da aplicação. Da 1ª versão do *Sputnik* pude retirar os seguintes dados:

- **5** *dashboards* com circuitos aplicacionais, por exemplo recargas e serviços da TMN (*wireless*) ou circuitos de facturação da PT (*wireline*);
- **207** sondas distribuídas pelos vários *dashboards*;
- **322** monitorizações que compõem as sondas;
- **805** *thresholds* para actualizar o estado das sondas;
- Cerca de **30.000** registos diários na BD;

Posteriormente, com a entrada da nova versão retirei os seguintes dados:

- **14** *dashboards*, para além dos antigos, foram adicionados novos, como o circuito BSS *Businness Support Service* da TT ou o circuito nacional de lojas da TMN;
- **673** sondas, que triplicaram em relação à primeira versão;
- **2.064** monitorizações, desde registos em BDs, à disponibilidade de servidores;
- **6.884** *thresholds*, cada vez mais adaptados a cada realidade para diminuir o erro;
- cerca de **180.000** registos diários na BD, dos quais cerca de **52.000** foram críticos;

Nos gráficos abaixo é possível ver a evolução da nova versão em comparação com a versão antiga:

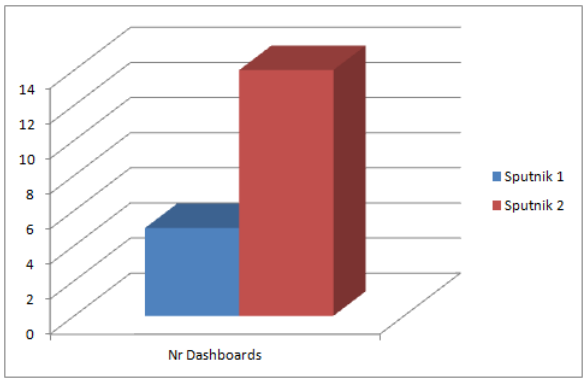


Figura 7.1: N.º Dashboards

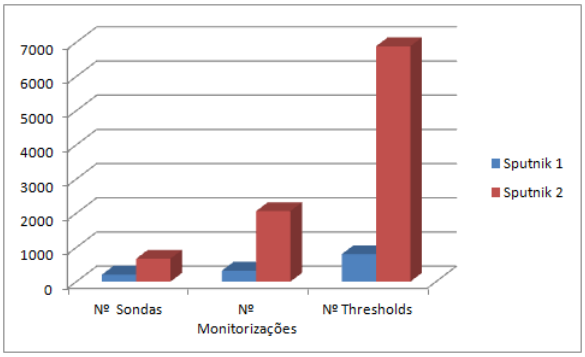


Figura 7.2: N.º Monitorizações

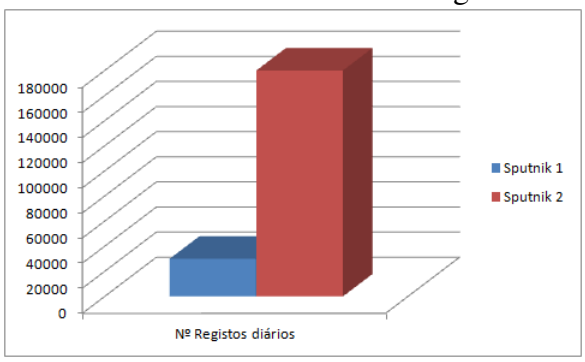


Figura 7.3: N.º Registos BD

Podemos constatar então, que os *dashboards* aumentaram **280%**, as sondas **325%**, as monitorizações **641%**, os *thresholds* **855%** e o número de registos na BD, diários, aumentou **600%**. Este aumento significativo é uma pequena amostra da boa escalabilidade do sistema, no entanto, apenas com a continuação do crescimento do *Sputnik*, poderei fazer uma análise cuidada da sua escalabilidade e a necessidade de remodelação do sistema.

#### 7.1.1.2 Trabalho Futuro

O *Sputnik* tem ainda um elevado potencial a nível de desenvolvimento, onde destaco, a noção de *threshold* dinâmico e uma área de indicadores gráficos.

O *threshold* dinâmico tem como objectivo tornar a aplicação o mais auto-suficiente possível e fiável para os utilizadores. Esta alteração permite que os valores mínimos e máximos dos *thresholds* sejam alterados dinamicamente consoante a altura do dia. Esta alteração é possível após analisar o histórico das monitorizações ao longo do tempo na tentativa de encontrar um comportamento padrão que gere alarmes falsos. A situação que deu esta ideia foi o facto de durante a madrugada recebermos dezenas de alarmes sobre a directoria de entrada de carregamentos da TMN estar vazia, por existirem poucos clientes a carregar o telemóvel.

Os indicadores gráficos permitirão perceber qual a performance dos processos, percebendo quais são os mais críticos e quais os pontos de estagnação de circuitos aplicacionais.

Além destas alterações, também poderei criar novos tipos de sondas, de maneira a poder alargar os horizontes do *Sputnik* a outras áreas e domínios e poderão existir mais integrações com outras aplicações internas ou mesmo externas, à OA.

### 7.1.2 Gateway Asterisk

Este projecto tem sido um sucesso junto das equipas que já desfrutam do seu serviço. Inicialmente pensou-se em criar apenas um piloto não só devido à falta de conhecimento da plataforma, pois não tinha a noção do tempo que iria demorar a montar o sistema, mas principalmente devido a um grande cepticismo por parte de equipas fora da DE/OA.

Como tal, o modelo de dados foi desenvolvido com base nos alarmes que Produção OA recebe via *HP-OVO* e portanto demasiado especifico para uma grande escalabilidade mas o suficiente para ter passado da fase de piloto para aplicação utilizada para reportar alarmes a várias equipas.

A integração com a *Mon-IT* mostrou-se bastante célere, dado que ambas as aplicações foram desenvolvidas no seio da minha equipa. Para além disso, a *Mon-IT* já funciona conjuntamente com o *HP-OVO*, permitindo assim adequar-se ao modelo de dados já existente.

Para a integração como outras aplicações, penso que será necessário uma revisão do modelo de dados, pois o funcionamento após a obtenção da informação para efectuar a chamada é genérico o suficiente para outro tipo de aplicações.

Apesar desse trabalho adicional que estou à espera, é espectável que a aplicação venha a ser utilizada fora da DE/OA e muito provavelmente inserida noutros domínios de utilização.

#### 7.1.2.1 Resultados

Desde que aplicação entrou em produção a meio de Abril, consegui retirar alguns dados estatísticos, como invocados do *Webservice*, chamadas automáticas efectuadas e número de alarmes mapeados:

- Existem **25** alarmes mapeados, de **4** equipas;
- O *Webservice* já foi invocado pelo *HP-OVO* **28.049** vezes, das quais resultaram **516** chamadas automáticas a reportar alarmes críticos;

Cada chamada tem 5 tentativas, efectuadas de 2 em 2 minutos, isto implica que no pior dos casos a chamada é atendida ao fim de 10 minutos. Até agora existiram **516** chamadas, o que equivale a **86** horas gastas neste processo. Isto equivale em *FTE* (*Full Time*

*Equivalent*), a **0.57 FTE** poupados, sendo que *1 FTE* equivale a **152** horas de trabalho de um recurso, por mês. Esta poupança de recursos pode traduzir-se em investimento de esforços noutras áreas carenciadas. Numa altura de cortes orçamentais e contenção, isto pode ser uma solução.

Como se pode ver no gráfico abaixo, o número de chamadas efectuadas pela plataforma, entre Abril e Maio, teve um crescimento de **417%**.

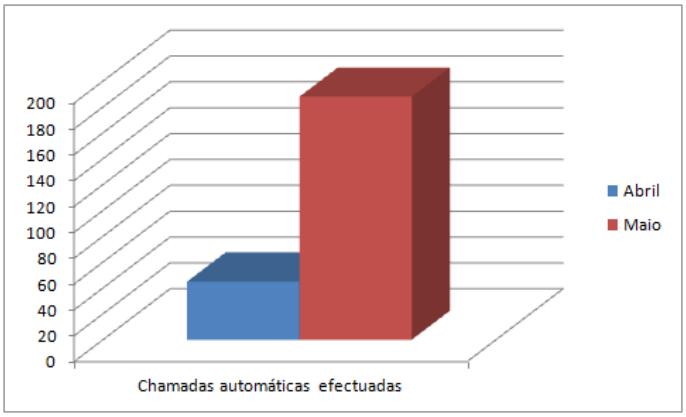


Figura 7.4: N.º chamadas automáticas

É expectável que até ao final do ano todos os alarmes, que indicam apenas para contactar a equipa de prevenção, estejam mapeados na aplicação, pois esta plataforma tem um grande potencial de poupança de recursos e registos de tempos de resposta mais baixos.

#### 7.1.2.2 Trabalho Futuro

Este projecto passou de *proof-of-concept* para um projecto totalmente em produção e portanto como trabalho futuro vou generalizar o modelo de dados da aplicação.

Esta generalização será uma forma de conseguir receber alarmes e despoletar chamadas automáticas sem ter em conta a aplicação de onde recebo a informação. Desta forma conseguirei uniformizar as invocações ao *Webservice* minimizando mais alterações futuras na estrutura da aplicação. Também pretendo melhorar a disposição da informação na aplicação *Web* e criar um processo de *Housekeeping* para as tabelas das chamadas e alarmes recebidos. O modelo de dados já contempla uma tabela de *backup* dos alarmes mas o processo ainda está em desenvolvimento.

Para além destes desenvolvimentos pretendo analisar os alarmes que as equipas de produção recebem e fazer o levantamento de todos os que implicam apenas um contacto telefónico. Depois deste levantamento, pretendo incentivar as equipas de suporte técnico a adoptarem os *reports* por chamada automática em vez de ser um elemento da produção a efectuar o contacto.

Poderá, também, fazer sentido uma integração com a *Checklist*, ficando assim o *Asterisk* responsável pela obtenção da informação sobre as equipas a contactar em caso de

alarme.

Finalmente, pretendo também vender esta ideia a outras áreas da PT e tentar aplicá-la noutros tipos de contextos, domínios e utilizações.

### 7.1.3 Checklist

A *Checklist* chegou a ser considerada o "*D. Sebastião da OA*"pois, na altura que entrei na empresa já se falava na criação de uma aplicação do género, mas na mesma altura começou a ser feita a *CMDB* corporativa e pensámos que iria corresponder às nossas exigências. O problema é que muita informação que existia na CMDB encontrava-se desactualizada e o facto de não podermos actualizar/inserir informação na mesma foi determinante para desenvolvermos o nosso projecto interno.

Neste momento temos cerca de **80%** da informação considerada indispensável para as equipas de produção da OA, já inserida da BD.

As integrações com a *Mon-IT* e *Medusa* têm tido um enorme sucesso dentro da OA pois a aquisição da informação necessária tornou-se simples e de rápido acesso. Sendo a aplicação central da OA, é espectável que a mesma tenha um crescimento acentuado em informação contida, consultas e integrações com outras aplicações internas à OA. Este acumular de carga será um bom teste à estabilidade e escalabilidade da aplicação.

#### 7.1.3.1 Resultados

Neste momento a *Checklist* é utilizada diariamente, há sensivelmente 1 semana, pelas equipas de produção da OA e já foi possível retirar os seguintes resultados:

- Existem **12** propriedades distribuídas por **9** tipos de CI;
- Existem **4** tipos de relações;
- **30** tipos de relações permitidas entre os tipos de CI;
- Já estão **500** CIs, dos quais **97** são máquinas, **154** são pessoas que pertencem a **31** equipas e 218 são aplicações, processos, entre outros;
- A aplicação é acedida para consulta cerca de *200* vezes por dia;

Com a continuação da utilização por parte das equipas da OA e a integração com novas aplicações internas, poderei verificar que a criação de um modelo de dados genérico permite uma enorme escalabilidade da aplicação.

#### 7.1.3.2 Trabalho Futuro

Como trabalho futuro a nível de desenvolvimento prevejo poucas alterações a nível da aplicação *Web*, mas no *Webservice* é diferente pois terei de criar novos métodos para as exigências propostas pelas aplicações que necessitam da informação. Tentarei sempre que cada *webservice* consiga comunicar com o máximo de aplicações possíveis, mas devido a especificações de algumas aplicações, terão de ser criados novos *webservices*.

Todas as integrações futuras e manutenção da informação na *Checklist* será efectuada por uma *Task force* que, também terá a responsabilidade de integrar a *Checklist* com outros processos e aplicações.

## 7.2 Lições

O meu percurso no PEI foi sempre em crescendo, conseguindo mesmo exceder as minhas expectativas pela rapidez da minha evolução pessoal e profissional no ataque aos desafios propostos. Consegui consolidar conhecimentos em linguagens com o *shellscript*, ou *SQL* e adquiri novos conhecimentos noutras linguagens como *C#* e *ASP.Net*.

Quando me propus a fazer os três projectos que constituem o meu PEI tinha apenas noções básicas de linguagens da *Microsoft* e a nível de *Web* tinha apenas a minha experiência académica, para além de todo o meu percurso académico ter sido mais focado na área de Arquitectura, Sistemas e Redes de Computadores por me identificar mais do que Sistemas de Informação. Nestes meses de PEI adquiri novas competências técnicas e uma visão sobre a área de Sistemas de Informação que não tinha até aqui, acabando por me entusiasmar na absorção de novos conhecimentos.

O facto de fazer o PEI fora o ambiente académico deu-me uma perspectiva completamente nova do que estava habituado, permitindo-me atingir uma maturidade pessoal e profissional que num ambiente académico não conseguia obter. Consegui consolidar os meus conhecimentos e evolui tecnicamente, pessoalmente e socialmente devido a todo o trabalho que desenvolvi e às responsabilidades que me foram acrescendo ao longo deste tempo.

Todos os projectos foram analisados e desenvolvidos minuciosamente de acordo com a análise de requisitos e objectivos propostos. Passaram por várias fases de desenvolvimento e prototipagem, sendo que a granularidade foi aumentando de etapa para etapa. Após conclusão das aplicações, passaram por uma etapa de testes rigorosos e finalmente entraram em produção. Utilizei, também, esta forma de planeamento dos projectos como uma forma de mitigação de riscos pois, consegui uma diminuição do risco de erros graves serem detectados apenas numa fase final.

Pessoalmente, consegui atingir vários objectivos pessoais, como evolução técnica, mas evolui principalmente numa vertente que nunca me tinha dedicado muito, que foi a parte de gestão. A área de gestão acabou por se tornar um objectivo pessoal, conciliando-

se também com os objectivos da minha equipa e da própria empresa. Todas as horas dispendidas em formações e no acompanhamento dos meus colegas ajudaram-me a evoluir no campo da gestão do tempo, pois tive de conciliar tudo durante o horário de trabalho.

Dentro da DE/OA existe ainda muito trabalho a ser desenvolvido no seio das equipas em que cada elemento deve proactivamente ajudar e melhorar o seu trabalho. Na minha equipa (DE/OA/MA) existe, também, muito caminho a percorrer pois, somos uma equipa recente e a primeira de desenvolvimento dentro da OA. Por isso, devemos continuar a analisar e automatizar processos, melhorando assim a resposta das equipas e desenvolver novas plataformas para facilitar o trabalho global da área.

No seguimento do impacto positivo e reconhecimento que este PEI está a ter na PTSI, pretendendo continuar numa senda de sucesso que me permita ganhar (ainda) maior visibilidade a nível profissional e poder evoluir cada ver mais, através de novos desafios e metas a alcançar.

# Acrónimos

| | |
|---|---|
| **ACD** | Automatic Call Direction |
| **API** | Application Programming Interface |
| **ASP** | Active Server Page (Microsoft script engine) |
| **ATM** | Automated Teller Machine |
| **BD** | Base de Dados |
| **BSM** | Business Service Management |
| **BSS** | Business Support Systems |
| **CI** | Configuration Item |
| **CMDB** | Configuration Management Database |
| **CRM** | Customer Relationship Management |
| **CSV** | Comma-Separated Values |
| **CTI** | Computer Telephony Integration |
| **DDL** | Data Definition Language |
| **DE** | Direcção de Exploração |
| **DML** | Data Manipulation Language |
| **E2E** | End-to-end |
| **ETL** | Extract-Transform-Load |
| **FTE** | Full Time Equivalent |
| **FTP** | File Transfer Protocol |
| **GSM** | Global System for Mobile Communications |
| **GUI** | Graphical User Interface |
| **HP** | Hewlett-Packard |
| **HTML** | HyperText Markup Language |
| **I/O** | Input/Output |
| **IDS** | Informix Dynamic Server |
| **IP** | Internet Protocol |
| **IS** | Information Systems |
| **IT** | Information Technologies |
| **ITIL** | Information Technology Infrastructure Library |

**JADE** Java Agent Development Framework
**JSON** JavaScript Object Notation
**JVM** Java Virtual Machine

**KPI** Key Performance Indicator

**LINQ** Language-Integrated Query
**LS** Linha de Serviço

**MA** Monitorização Aplicacional
**MB** Multibanco
**MVC** Model-view-controller

**NAT** Network Address Translation

**OA** Operação Aplicacional
**OOTB** Out of the box
**OS** Operating System
**OSS** Operational Support Systems

**PBX** Private Branch Exchange
**PHP** Hypertext Preprocessor
**POC** Proof-of-concept
**PPS** Prepaid Service
**PT** Portugal Telecom
**PTSI** Portugal Telecom Sistemas de Informação

**QoS** Quality of Service

**RAM** Random Access Memory
**RCA** Root Cause Analysis
**RFC** Request For Change
**ROI** Return On Investment

**SGBD** Sistema de Gestão de Base de Dados
**SH** Shell Script
**SI** Sistemas de Informação
**SIBS** Sociedade Interbancária de Serviços
**SIP** Session Initiation Protocol
**SL** Semantic Language
**SLA** Service Level Agreement
**SLO** Service Level Objective
**SMS** Short Message Service
**SO** Sistema Operativo

**SP** Storage Procedure
**SSH** Secure Shell

**TCO** Total Cost of Ownership
**TI** Tecnologias de Informação
**TMN** Telecomunicações Móveis Nacionais

**UML** Unified Modelling Language
**URI** Uniform Resource Identifier
**URL** Uniform Resource Locator

**VBS** Visual Basic Script
**VM** Virtual Machine
**VOIP** Voice over IP
**VuGen** Virtual User Generator

**WBS** Work Breakdown Structure
**WSDL** Webservice Description Language
**WWW** World Wide Web

**XHTML** Extensible Hypertext Markup Language
**XML** Extensible Markup Language

# Bibliografia


[1] Jr Adam Freeman, Joseph C. Rattz. *Pro LINQ: Language Integrated Query in C# 2010*. Apress, 2010.

[2] Wolfgang Barth. *Nagios: System and Network Monitoring*. No Starch Press, 2008.

[3] Douglas Crockford. *JavaScript: The Good Parts*. O'Reilly Media, 2008.

[4] Inc Digium. Asterisk wikipedia. https://wiki.asterisk.org/wiki/display/AST/Home.

[5] Jim Wooley Fabrice Marguerie, Steve Eichert. *LINQ in Action*. Manning Publications Co., 2008.

[6] David Flanagan. *JavaScript: The Definitive Guide, 4th Edition*. O'Reilly Media, 2001.

[7] John C. Goodpasture. *Quantitative Methods in Project Management*. J. Ross Publishing, Inc., 2009.

[8] Leif Madsen Jim Van Meggelen and Jared Smith. *Asterisk The Future of Telephony*. O'Reilly Media, Inc., 2nd edition, 2007.

[9] Jim Van Meggelen Leif Madsen and Russell Bryant. *Asterisk: The Definitive Guide*. O'Reilly Media, Inc., 2011.

[10] APM Group Ltd. Official itil website. http://www.itil-officialsite.com/.

[11] MediaWiki. Latex wikipedia. http://en.wikibooks.org/wiki/LaTeX.

[12] Cabinet Office. *ITIL Service Design*. TSO (The Stationery Office), 2011.

[13] UK Cabinet Office. Best management practice portfolio. http://www.cabinetoffice.gov.uk/resource-library/best-management-practice-portfolio.

[14] Scott Pakin. The comprehensive latex symbol list. *Comprehensive TEX Archive Network*, September 2003.

[15] Mike Peckar. *Fognet's Field Guide to OpenView Network Node Manager*. Fogbooks, 2006.

[16] Joseph Phillips. *PMP Project Management Professional Study Guide, Third Edition*. McGraw-Hill Prof Med/Tech, 2009.

[17] Agent Oriented Software Pty. Jack intelligent agents agent manual. http://www.aosgrp.com/documentation/jack/Agent_Manual_WEB/index.html.

[18] Johannes Gehrke Raghu Ramakrishnan. *Database Management Systems*. McGraw-Hill, 3rd edition, 2003.

[19] Chris Gallelli Ray Rankins, Paul Bertucci and Alex T. Silverstein. *Microsoft SQL Server 2008 Unleashed*. McGraw-Hill Osborne Media, 1st edition, 2011.

[20] John Resig. *Pro JavaScript Techniques*. Apress, 2006.

[21] Steven Sanderson. *Pro ASP.NET MVC V2 Framework (Expert's Voice in .NET)*. Apress, 1nd edition, 2010.

[22] Steven Sanderson. *Pro ASP.NET MVC V3 Framework*. Apress, 3nd edition, 2011.

[23] Moraitis P. Spanoudakis N. Modular jade agents design and implementation using aseme. 2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (IAT'10), 2010.

[24] Katia Sycara-Cyranski. *The RETSINA MAS Infrastructure*. Carnegie Mellon University, the Robotics Institute, 2001.

[25] Paul Weber Tammy Zitello, Deborah Williams. *HP OpenView System Administration Handbook: Network Node Manager, Customer Views, Service Information Portal, OpenView Operations*. Prentice Hall, 2004.

[26] Hyong S. Kim Tiago Carvalho. Etymon: A root cause analysis system for large and complex it enterprise networks. *[PFARM'09] IEEE/IFIP DSN Workshop on Proactive Failure Avoidance, Recovery, and Maintenance (PFARM)*, June 2009.

[27] Andrew Troelsen. *Pro C# 2008 and the .NET 3.5 Framework*. Springer, 4rd edition, 2007.